

Reinforcement Learning for Solving Financial Problems

by

Lufan Wang

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2024

© Lufan Wang 2024

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This thesis explores the application of reinforcement learning (RL) to address two important financial problems: risk management and optimal trade execution.

In risk management, we aim to balance returns with associated risks. To achieve this, we propose an enhanced RL model that integrates a dynamic Conditional Value at Risk (CVaR) measure. By leveraging distorted probability measures, CVaR allows the RL agent to emphasize worst-case scenarios, ensuring that potential losses are accounted for while optimizing long-term returns. Our method substantially reduces the model's training time by efficiently reusing computation results, significantly lowering computational overhead. Furthermore, it optimizes the balance between exploration and exploitation. This approach leads to more robust decision-making in uncertain environments and a better overall return.

For optimal trade execution, we formulate a flexible RL-based framework capable of dynamically adjusting to changing market conditions. Our model not only replicates the results of Almgren-Chriss model in linear environments but also demonstrates superior performance in more complex, nonlinear scenarios where traditional methods like Almgren-Chriss face challenges.

Acknowledgements

First, I would like to express my deepest gratitude to my supervisor, Justin Wan, for his invaluable guidance and support throughout my master's program and the thesis process. His mentorship has profoundly shaped my academic growth, and I am confident that the lessons I've learned from him will benefit me throughout my life. I consider myself incredibly fortunate to have had him as my supervisor.

I would also like to extend my sincere thanks to my readers, Professor Yaoliang Yu and Professor Kimon Fountoulakis, for taking the time to review my thesis and providing insightful feedback. Your thoughtful input has been invaluable in shaping the final version of this work.

I am deeply grateful to my family and friends for their unwavering encouragement and support. Your presence, both near and far, has been a constant source of strength, and I could not have completed this journey without your emotional support.

Lastly, I want to express my heartfelt thanks to my husband, Zhangyuan Nie. The past years, particularly the last six months, have been an especially challenging time for me. Your boundless patience, understanding, and generosity have helped me through it all. I am truly grateful for your love and support.

Dedication

This is dedicated to the one I love and the one who loves me.

Table of Contents

Author's Declaration	ii
Abstract	iii
Acknowledgements	iv
Dedication	v
List of Figures	ix
List of Tables	x
1 Introduction	1
1.1 Risk Management	1
1.2 Optimal trade execution	3
1.3 Thesis Outline	4
2 Background	5
2.1 The RL Model	5
2.2 Markov Decision Process	7
2.3 Cart-Pole Balancing	9
2.4 Optimization in Reinforcement Learning	10

2.4.1	Value-Based Methods	10
2.4.2	Policy-Based Methods	11
2.5	Actor-Critic Algorithm	12
2.6	Other RL Algorithms	13
3	Dynamic CVaR Risk Measure	14
3.1	The risk measure setting	14
3.1.1	VaR and CVaR	15
3.1.2	Reformulation of CVaR	16
3.1.3	Dynamic convex risk measures	18
3.2	The RL algorithm setting	23
3.2.1	Risk-averse objective function	23
3.2.2	Optimization problem formulation	24
3.2.3	Improvement	27
3.3	The RL algorithm	30
4	Optimal Trade Execution	34
4.1	Introduction	34
4.2	Problem Formulation	35
4.3	Almgren-Chriss Model	37
4.4	The RL setting	40
4.4.1	Action	40
4.4.2	State	40
4.4.3	Transition Probability	41
4.4.4	Reward	41
4.4.5	Counter	45
4.5	The RL algorithm	46

5	Experiment	50
5.1	Dynamic CVaR Risk measures on RL application	50
5.1.1	Statistical Arbitrage	50
5.1.2	Jump Cases	57
5.1.3	Stock Trading	64
5.2	Optimal Trade Execution	68
5.2.1	Optimal Execution of Portfolio Transactions	69
5.2.2	Nonlinear Price Impact Function	74
6	Conclusions and Future Work	76
6.1	Conclusions	76
6.2	Future Work	77
	References	79

List of Figures

2.1	The interaction process in reinforcement learning	6
5.1	Terminal wealth distributions for training loops $l = 300, 1200$ with $\psi = 0.05$	53
5.2	Terminal wealth distributions for $N = 6, 11, 21$ with $\psi = 0.05$	55
5.3	Terminal wealth distributions for $N = 6, 11, 21$ with $\psi = 0$	56
5.4	Performance of the model trained with $\lambda = 0$	59
5.5	Performance of the model trained with $\lambda = 0.005$	60
5.6	Performance of the model trained with $\lambda = 0.05$	61
5.7	Performance of the model trained with $\lambda = 0$	63
5.8	Performance of the model trained with $\lambda = 0.005$	64
5.9	Performance of the model trained with $\lambda = 0.05$	65
5.10	Comparison of distribution performance between our model and the baseline across different jump scenarios	68
5.11	Efficient frontier curves for each λ value.	70
5.12	$\min(E(x) + \lambda \cdot \text{Var}(x))$ for each test λ case.	71
5.13	Relative error of $E(x) + \lambda \cdot \text{Var}(x)$ for each test λ case.	71
5.14	Shares Holding vs Time Periods for each test lambda cases.	72
5.15	Efficient frontier curves for each λ value.	74
5.16	$\min(E(x) + \lambda \cdot \text{Var}(x))$ for each test λ case.	75

List of Tables

5.1	Parameter values used for example 5.1.1.	52
5.2	Comparison of Statistics for training loops $l = 300, 1200$ with $\psi = 0.05$. .	53
5.3	Comparison of Statistics for $N = 6, 11, 21$ with $\psi = 0.05$	54
5.4	Comparison of Statistics for $N = 6, 11, 21$ with $\psi = 0$	56
5.5	Comparison of Statistics for $lambda = 0, 0.005, 0.05$ with jumps in $[0.05, 0.1]$	58
5.6	Comparison of Statistics for $\lambda = 0, 0.005, 0.05$ with jumps in $[0.03, 0.06]$. .	62
5.7	Parameter values used for example 5.1.3.	66
5.8	Comparison of Statistics for Various Scenarios	67
5.9	Parameter values used in the optimal trade execution example.	69
5.10	Comparison of Time Periods to Stop for Our Results and Optimal Results for different λ Values	73

Chapter 1

Introduction

The finance industry is one of the most influential sectors globally, given its crucial role in shaping economies and facilitating wealth management[6]. There exist various types of financial problems, such as hedging[45], stock trading[51], and pension planning[36, 17]. Among the numerous financial issues in the world, two have particularly drawn our interest. The first is risk management[2], a fundamental aspect of finance consistently emphasized due to its critical importance in safeguarding assets and ensuring financial stability. The second is the optimal trade execution problem[8, 3], which focuses on executing trades efficiently to maximize returns under specific variance constraints. In this thesis, I will focus on applying reinforcement learning techniques to address these two financial problems.

1.1 Risk Management

Investing inherently involves balancing returns with risk[30]. While higher returns can lead to greater profits, they are often accompanied by increased risk, which raises the likelihood of losses and leads to more volatile outcomes. Conversely, strategies that minimize risk tend to reduce uncertainties in trading but typically result in lower returns. Therefore, finding a balance between return and risk is essential in finance[31]. Risk management plays a crucial role in optimizing this trade-off between risk and return[46]. It involves the process of identifying, assessing, and controlling risks that arise during the trading process. The core objective of risk management is to safeguard assets and ensure financial stability by minimizing the impact of potential adverse events.

Risk measures serve as fundamental tools within the risk management framework, enabling the quantification and management of risk[32]. By tracking and understanding

quantified risks, traders can make informed decisions on whether to accept, mitigate, or transfer risks. Adapting risk measures to solving financial problems helps ensure that risks remain within acceptable limits and that the strategies implemented are robust.

With the rapid advancement of technology, machine learning has emerged as a powerful tool in financial risk management[25, 4]. Unlike human traders, machine learning models can analyze and respond to scenarios within milliseconds, operating continuously without the need for rest. By incorporating risk measures into the training process, a machine learning agent continuously learns from the insights these measures provide into the risk, developing robust strategies similar to how a human trader would refine their approach over time[4]. Among various machine learning approaches, reinforcement learning (RL) has drawn our attention. RL is a major branch of machine learning that has played a significant role in various applications, such as computer vision[50], medical image analysis[53], and natural language processing[28]. A key advantage of RL is that it does not require direct instructions or labeled data, unlike supervised learning, which depends heavily on predefined examples for training[47]. In real-world scenarios, especially in financial markets, it is impractical to cover all possible market conditions. RL addresses this challenge by allowing the model to explore and learn from its own experiences, adapting to new and unforeseen situations without needing explicit guidance. During the training process, RL models learn by taking actions that not only affect the current state but also influence future states and outcomes. The objective of RL is to minimize long-term risks rather than focusing solely on immediate losses. This broader perspective mirrors the approach required in finance, where the ultimate goal is often to optimize cumulative costs over time.

All of the above mentioned characteristics of RL technology align well with financial problems, making it an excellent approach for addressing complex economic challenges. Several studies have incorporated risk measures into RL, addressing complex decision-making in various domains. The major approaches for reinforcement learning integrated with risk measures include mean-variance optimization[22], which balances expected returns with their variability, and distributional RL[13], which focuses on learning the entire distribution of potential returns to better handle risk.

In finance, the RL learner focuses on minimizing financial losses using measures such as Conditional Value at Risk (CVaR), which is crucial for applications like portfolio optimization and trading strategies. In [11], the researchers proposed an approach adapting percentile-based risk measures, such as CVaR, within their RL framework, enabling agents to control worst-case outcomes while learning in uncertain environments. The study in [12] introduced the integration of dynamic risk measures into the evaluation of optimization problems, reformulating time-consistent risk-sensitive optimization within a reinforcement learning framework. Their work focused on assessing the risk of sequences of uncertain

outcomes arising from market scenarios. These risk measures helped guide the RL agent in identifying optimal trading strategies for financial decision-making under uncertainty, effectively balancing risk and return in a more adaptive and comprehensive way. Expanding on [12], we propose an enhanced RL model with a dynamic CVaR risk measure, offering the following advantages:

(i) Reuse of policy network results in value network evaluation: This approach not only enhances the efficiency of the learning process by reducing redundant calculations but also strengthens the consistency between policy decisions and value assessments across timesteps, thereby leading to a more stable and connected learning trajectory.

(ii) Improved balance between exploration and exploitation in RL: By maintaining exploration in the learning process while focusing on the best outcomes during value evaluation, our approach achieves a better average performance, leading to more optimal final results.

1.2 Optimal trade execution

The optimal trade execution problem is a critical issue in finance, where the objective is to execute large trades in a way that minimizes costs and market impact while maximizing returns within a certain timeframe. The preliminary models were introduced by [8, 3], determining optimal trading strategies through dynamic programming. Since then, researchers have explored various alternative approaches, including PDEs [16] and game theory [15]. Despite their differing techniques, these methods share a common limitation: they struggle to adapt when introducing new parameters, handling nonlinear conditions, or extending to multiple assets.

In contrast, reinforcement learning (RL) offers a flexible framework where an agent learns through interaction with the trading environment. The training process in RL is driven by the agent’s continuous interaction with the environment. As the environment changes, the RL model adapts by learning from new experiences, allowing it to adjust its strategy dynamically based on updated conditions. This adaptability makes RL particularly well-suited for complex scenarios. Previous researches [27, 20] have explored the application of reinforcement learning (RL) to optimal trade execution problems. In [27], the reward signal was omitted during intermediate timesteps, focusing only on the final reward at the end of the trading period. This approach, however, may ignore the importance of intermediate actions in shaping the final outcome, as the actions taken during these steps have a cumulative effect on the overall performance. In contrast, the study by

Hendricks and Wilcox [20] demonstrated that RL models without considering incorporating the variance into their rewards, leading to a higher execution risk though they get a higher expected returns. These drawbacks can lead to suboptimal learning and decision-making during the trade execution process.

In this paper, we propose an enhanced RL framework for solving the optimal trade execution problem, addressing the limitations of previous models and achieving performance comparable to that of [3]. Our RL framework offers the following key advantages:

(i) The model is inherently model-free and does not require prior knowledge of the optimal solution or a specific model structure. This allows it to learn and adapt dynamically to a wide range of complex trading scenarios, including those that pose challenges for traditional methods like [3].

(ii) By directly deriving the optimal strategy through its interactions with the trading environment, the RL approach eliminates the need for dynamic programming and avoids potential related error transmission. This flexibility makes it easier to scale to higher-dimensional problems, where multiple diverse assets may be involved.

(iii) In addition to focusing on returns, our model also considers the variance of trade execution in the reward formulation. By incorporating execution risk directly into the learning process, the agent learns to balance both risk and return, resulting in a more robust trading strategy that minimizes risk while maximizing returns.

(iv) Our RL model takes into account the impact of intermediate steps on the final outcome. This means the agent is rewarded based on the cumulative effects of its actions, rather than assigning a zero reward to middle steps, leading to a more accurate optimization of the final trading performance.

1.3 Thesis Outline

The rest of this thesis is organized as follows: Chapter 2 provides a brief introduction to the background of reinforcement learning. Chapter 3 explores risk measurement using RL technology and discusses enhancements in this application. Chapter 4 presents an RL approach to solving the optimal trade execution problem. Chapter 5 describes the experiments conducted using the methodologies outlined in Chapters 3 and 4, along with a presentation of the results. Finally, Chapter 6 concludes the thesis and discusses potential future research directions.

Chapter 2

Background

In this chapter, we review the foundational concepts of reinforcement learning (RL), building on the work of [48]. We start by introducing the basic structure of the RL model and the Markov Decision Process (MDP), followed by a simple illustrative example. Next, we explore common learning approaches, with particular emphasis on the actor-critic algorithm, which will be central to the subsequent chapters. Finally, we provide a brief overview of other well-known RL algorithms.

2.1 The RL Model

In an RL model, we usually treat the RL learner as the *agent*, and the thing it interacts with is defined as the *environment*. The agent learns optimal policies through interactions with the environment. Specifically, for a sequence of discrete timesteps, $t = 0, 1, 2, \dots$, at each step of interaction, the agent receives the current state $s_t \in \mathbf{S}$, where \mathbf{S} is an arbitrary state space containing all possible states, and a reward r_t obtained from the reward function \mathbf{R} . The state s_t represents whatever information is available to the agent. The agent then selects an action $a_t \in \mathbf{A}$, where \mathbf{A} is an arbitrary action space containing all possible actions. The environment receives the input a_t and provides feedback for the current timestep t , which consists of the reward r_t and the next state s_{t+1} . Notably, $r_t = \mathbf{R}(s_t, a_t, s_{t+1})$ indicates the reward received when the agent is in state s_t , takes action a_t , and transitions to state s_{t+1} . A diagram of the entire interaction process is shown in Figure 2.1.

Besides the agent and the environment, there are three more components needed to build a reinforcement learning model: the goal, the policy π , and the value function V .

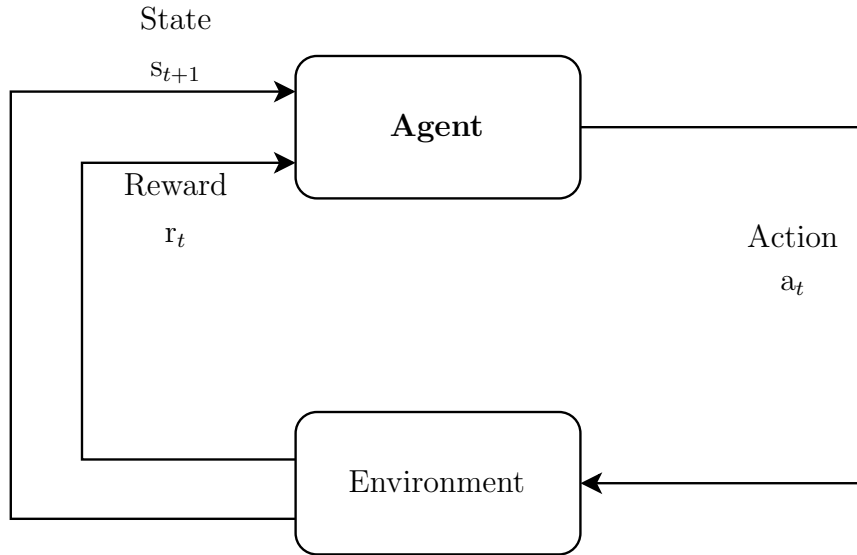


Figure 2.1: The interaction process in reinforcement learning

The objective of *risk-neutral* reinforcement learning is to maximize cumulative rewards, commonly referred to as maximizing total returns. Although the reward r_t depends on the current state s_t , the action a_t , and the subsequent state s_{t+1} , the agent only has control over the action a_t . By selecting a_t , the agent directly influences the rewards it receives. If the reward is high, the action is considered favorable; otherwise, the agent may need to explore alternative actions. The agent can learn to develop an effective strategy for selecting actions by evaluating the reward values. However, the objective of an RL agent can vary depending on the specific problem. For example, in a *risk-averse* setting, the agent's goal may shift from maximizing rewards to minimizing risk. For the remainder of this chapter, we will focus exclusively on a risk-neutral RL model.

A policy π is a strategy or rule that an agent follows to decide which action to take in a given state. The optimal policy π^* selects actions to maximize cumulative rewards over time. Policies can be either deterministic or stochastic. A deterministic policy always selects the same action for a given state. It is represented mathematically as a function:

$$\pi(s_t) = a_t.$$

This means that for any state s_t , the policy π will always select action a_t . A stochastic policy, however, specifies a probability distribution over possible actions for each state. In a stochastic policy, $\pi(a_t | s_t)$ represents the probability of taking action a_t when in state

s_t . As the agent may choose different actions in the same state based on the probabilities, this randomness encourages the agent to explore different actions, thereby increasing the chances of discovering more effective strategies. By preventing the agent from becoming stuck in suboptimal strategies, stochastic policies are often preferred in reinforcement learning algorithms. For the remainder of this thesis, whenever we refer to a policy, we will be considering a stochastic policy.

As the goal is to maximize cumulative rewards, for a given state s_t , the agent must consider not only the immediate reward r_t from action a_t but also the future returns that this action may yield. For instance, if the agent consistently receives a low immediate reward at state s_t , but transitions to a future state s_{t+1} where the expected rewards are significantly higher, state s_t would still be considered favorable as it leads to better outcomes in the long run, resulting in higher cumulative rewards. The value of a state s_t is determined by the expected cumulative rewards the agent can obtain starting from s_t . The value function V takes a state as input and outputs the quantified value of that state. The value function V is expressed as:

$$V(s_t) = \mathbb{E} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t \right], \quad (2.1.1)$$

$\gamma \in [0, 1]$ here is used as a discount factor to control how much future rewards are valued compared to immediate ones. A higher γ value indicates that future rewards are more significant, while a lower γ emphasizes immediate rewards. In extreme cases, when $\gamma = 1$, future rewards are considered equally important as present ones, whereas $\gamma = 0$ means the agent focuses solely on immediate rewards. In equation (2.1.1), γ^k represents the discount factor raised to the power of k , which reduces the value of rewards received k steps into the future, and r_{t+k} denotes the reward obtained at time $t + k$.

Optionally, there may be a fourth component in the system called the model of the environment. If the agent learns solely from direct interactions with the environment using trial and error, it is a model-free method. If the agent not only learns from trial and error but also tries to model the environment, it is a model-based method, which includes a model of the environment. In the following chapters, we will focus only on model-free RL methods.

2.2 Markov Decision Process

A fundamental concept in reinforcement learning is the *Markov Decision Process* (MDP)[5]. The MDP is a discrete-time stochastic control process widely used across various domains,

including reinforcement learning. Most importantly, it provides a mathematical framework for modeling decision-making situations.

A model is said to be *Markov* if the environment’s response at the next timestep depends only on the current state and the action taken. More specifically, if there exists a probability transition function $\mathbb{P}(s_{t+1} | s_t, a_t)$ that specifies the next state s_{t+1} from the environment as a function of its current state s_t and the action a_t , and it is independent of any previous environment states or agent actions, then the model is Markov[21].

A typical MDP framework used in RL is a tuple that contains elements defining the environment in which the agent operates. An MDP is represented as a 4-tuple $\langle \mathbf{S}, \mathbf{A}, \mathbb{P}, \mathbf{R} \rangle$ where each element is defined as follows:

- **S**: An arbitrary state space, describing a set of possible states an agent can reach. $\mathbf{S} \subset \mathbb{R}^d$ with $d \geq 1, d \in \mathbb{Z}$.
- **A**: An arbitrary action space, describing a set of possible actions an agent can take. $\mathbf{A} \subset \mathbb{R}^n$ with $n \geq 1, n \in \mathbb{Z}$.
- **P**: The transition probability function, is assumed to be stationary and unknown to the agent. It characterizes $\mathbb{P}(s_{t+1} = s' | s_t = s, a_t = a)$, the probability of transitioning to state s' from state s when action a is taken.
- **R**: The reward function $\mathbf{R} : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \rightarrow \mathbb{R}$, which specifies the immediate reward received after transitioning from state s to state s' due to action a .

Although general MDPs may have infinite (even uncountable) state and action spaces, in this paper, we consider only the finite MDP case. A finite MDP refers to an MDP where the decision-making process has a fixed number of decision steps, often denoted as T . The agent makes decisions over a finite time period, and after T steps, the process terminates. This whole process is called one episode. For an RL model with finite MDP, at each timestep $t \in [0, T - 1]$, the current policy π_t selects an action $a_t \in \mathbf{A}$ based on the current state $s_t \in \mathbf{S}$, leading to the next state s_{t+1} . The agent receives an immediate reward $r_t = R(s_t, a_t, s_{t+1})$ at each timestep t . The γ -discounted cumulative rewards for a trajectory from $t = 1$ to $T - 1$ under a policy $\pi = \{\pi_0, \pi_1, \dots, \pi_{T-1}\}$ is calculated as:

$$\max_{\pi} \mathbb{E} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right], \tag{2.2.1}$$

where $\gamma \in [0, 1]$ is the discount factor. The optimal strategy π^* is the one that maximizes equation (2.2.1). Equation (2.2.1) represents the objective function for risk-neutral RL. And the value function V in equation (2.1.1) can be further extended as:

$$\begin{aligned}
 V(s_t) &= \mathbb{E} \left[\sum_{k=0}^{T-t-1} \gamma^k r_{t+k} \mid s_t \right] \\
 &= \mathbb{E} \left[r_t + \gamma \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1} \mid s_t \right] \\
 &= \mathbb{E} [r_t + \gamma V(s_{t+1}) \mid s_t].
 \end{aligned} \tag{2.2.2}$$

Equation (2.2.2) is the Bellman equation for V . It shows a connection between the value of a state and the values of its successor states. The Bellman equation is often used in dynamic programming like Value Iteration [7] or Policy Iteration [7, 26].

2.3 Cart-Pole Balancing

A classic example of reinforcement learning with a finite MDP is the Cart-Pole Balancing Problem. In this scenario, the task is to balance a pole on a moving cart by applying force to either push the cart to the left or the right. The agent’s objective is to prevent the pole from falling over by keeping it balanced for as long as possible within a fixed time horizon T .

The state space \mathbf{S} in this problem is composed of four continuous variables: the position of the cart, the velocity of the cart, the angle of the pole, and the angular velocity of the pole. The action space \mathbf{A} contains only two possible actions: applying a force to push the cart to the left or the right. The transitions between states are governed by the physical dynamics of the system, and these transitions are captured by the transition probability function $\mathbb{P}(s_{t+1} \mid s_t, a_t)$, which determines how the system moves from one state to another based on the agent’s action and the physics of the cart and pole. For instance, if the agent chooses to push the cart to the left, this action will cause both the cart’s velocity and the pole’s angle to change based on the system’s physical dynamics. In terms of rewards, the agent receives a positive reward of +1 for every time step that the pole remains balanced. However, if the pole falls beyond a certain angle or the cart moves outside a predefined range, the episode ends, and the agent incurs a large negative reward (e.g., -20). Additionally, if the pole remains balanced for the entire duration of T steps, the episode also terminates. Since the RL agent’s goal is to maximize cumulative rewards,

it will aim to keep the pole balanced for as long as possible to accumulate more rewards while avoiding penalties incurred by the pole falling off.

Since the goal of the RL agent is to maximize cumulative rewards, it will aim to keep the pole balanced for as long as possible to accumulate more rewards while avoiding penalties incurred by the pole falling off. This aligns perfectly with the problem’s objective, as keeping the pole balanced longer is the desired outcome.

2.4 Optimization in Reinforcement Learning

In reinforcement learning, there are two primary approaches for finding the optimal policy that maximizes cumulative rewards: *value-based methods* and *policy-based methods*.

2.4.1 Value-Based Methods

In value-based methods, the agent learns the value of each state s_t , which represents the expected cumulative reward that the agent can achieve starting from that state. Once the agent has a good estimate of the value of each state, the agent selects actions that maximize the expected return from each state, ultimately leading to the optimal policy.

The value function V is parameterized by a set of parameters ϕ (e.g., the weights in a linear model or a neural network). During the training process, the parameters ϕ are adjusted to minimize the error between the estimated value $V(s_t)$ and the actual returns observed from the environment. Once this error falls below a specified tolerance, the current parameters ϕ are considered optimal, denoted as ϕ^* , which best approximate the value function for each state s_t over the time horizon $t \in [0, T - 1]$. This adjustment is controlled by a hyperparameter known as the learning rate α , which dictates the size of the updates made to the parameters during each step of training.

To enhance understanding of value-based methods, we present two examples: the *Monte Carlo* (MC) method and the *Temporal Difference* (TD) learning method. We will introduce them as follows.

Monte Carlo Method

In the Monte Carlo (MC) method[34], the value function is evaluated by simulating sequences of states, actions, and rewards (i.e., a trajectory $(s_0, a_0, r_0, \dots, s_{T-1}, a_{T-1}, r_{T-1})$)

during each episode. For each visit to state s_t , the return is calculated as:

$$\sum_{k=0}^{T-t-1} \gamma^k r_{t+k}.$$

The target value for $V(s_t)$ is then the average of all such returns across multiple visits to s_t . For a value function V parameterized by ϕ and with learning rate α , ϕ is updated by:

$$\phi \leftarrow \phi + \alpha (\text{Average of returns at } s_t - V(s_t)).$$

The Monte Carlo method requires complete episodes to compute the value of states or state-action pairs. This makes it unsuitable for environments where episodes are long or ongoing. However, it is a model-free approach, meaning it does not require knowledge of the environment's dynamics, making it effective in complex, unknown environments.

Temporal Difference (TD) Learning

Similar to the Monte Carlo method, Temporal Difference (TD) learning[49] does not require a complete model of the environment. However, instead of using full returns like the Monte Carlo method, the TD method applies the Bellman equation in equation (2.2.2) and updates the value function incrementally. The target value is the immediate reward plus the estimated value of the next state:

$$\text{TD Target} = r_{t+1} + \gamma V^\phi(s_{t+1}).$$

The TD error, which measures the difference between the estimated and actual returns, is calculated as:

$$\text{TD Error} = r_{t+1} + \gamma V^\phi(s_{t+1}) - V^\phi(s_t).$$

And ϕ is updated by:

$$\phi \leftarrow \phi + \alpha \text{TD Error}.$$

This allows TD learning to update the value function after each step, rather than waiting for an episode to complete, making it faster and more efficient.

2.4.2 Policy-Based Methods

Policy-based methods, in contrast to value-based methods, aim to directly optimize the policy without estimating a value function. The policy is represented as $\pi_\theta(a | s)$, where θ are the parameters of the policy. The objective is to adjust θ to maximize the expected cumulative reward by using gradient ascent on the policy parameters. This approach is known as the *Policy Gradient* method.

Policy Gradient Method

Recall that the goal of a reinforcement learning agent is to solve the optimization problem in equation (2.2.1). A common approach to achieve this is to perform gradient ascent on the objective function with respect to the policy parameters θ [48]. The objective function is given by:

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

The gradient of the objective function $J(\theta)$ is computed and used to update the policy parameters as follows:

$$\theta \leftarrow \theta + \beta \nabla_{\theta} J(\theta),$$

where β is the learning rate. This approach is particularly effective in continuous or high-dimensional action spaces, where deriving a policy directly from a value function can be challenging.

2.5 Actor-Critic Algorithm

Algorithm 1 Actor-Critic Algorithm

- 1: Initialize the value function network ϕ with learning rate α and the policy network θ with learning rate β .
 - 2: Initialize the environment and Adam optimizers.
 - 3: **for** each episode $l = 1, \dots$ **do**
 - 4: **for** each timestep t of the episode **do**
 - 5: Select action a_t according to the current policy $\pi^{\theta}(a_t | s_t)$.
 - 6: Perform action a_t , observe reward r_t and next state s_{t+1} .
 - 7: Compute TD error: $\delta_t = r_t + \gamma V^{\phi}(s_{t+1}) - V^{\phi}(s_t)$.
 - 8: Update the critic (value function): $\phi \leftarrow \phi + \alpha \delta_t$.
 - 9: Update the actor (policy): $\theta \leftarrow \theta + \beta \nabla_{\theta} J(\theta)$.
 - 10: **end for**
 - 11: **end for**
 - 12: **Done.**
-

The Actor-Critic algorithm, introduced in [24], is a hybrid approach that combines policy-based methods (actor) with value-based methods (critic) to improve both stability

and learning efficiency. The actor is responsible for selecting actions according to the policy, while the critic evaluates these actions by using the value function. By integrating the strengths of both approaches, the Actor-Critic algorithm achieves more stable and effective learning outcomes compared to using either value-based or policy-based methods alone. In this framework, we use the TD learning method for updating the critic and the Policy Gradient method for updating the actor.

The Adam optimizer[23] (short for Adaptive Moment Estimation) is widely used in RL due to its adaptive learning rate and efficient handling of noisy, sparse gradients, making it particularly suitable for training neural networks. Adam is commonly employed in Actor-Critic algorithms, where neural networks approximate both the policy (actor) and the value function (critic).

2.6 Other RL Algorithms

Besides the actor-critic algorithm, there are other popular RL algorithms. Although these algorithms are not used in our experiments, we provide a brief introduction to them here.

- **Soft Actor-Critic (SAC)**: An improved actor-critic algorithm that aims to maximize both the expected reward and the entropy of the policy to encourage exploration[19], it has quickly become one of the most popular RL algorithms due to outperforming prior on-policy and off-policy methods in sample efficiency and asymptotic performance.
- **Deep Q-Network (DQN)**[35]: An algorithm designed for discrete action spaces that combines Q-learning with deep neural networks.
- **Q-Learning**: A classic value-based algorithm where the agent learns the value of taking an action in a particular state using the Bellman equation.

Chapter 3

Dynamic CVaR Risk Measure

In this chapter, we focus on solving financial investment problems using RL. We begin by exploring the concept of risk measures, with particular emphasis on the Conditional Value at Risk (CVaR) risk measure. We then reformulate CVaR and construct a dynamic CVaR risk measure, drawing from the work of [39]. Following this, we build a *risk-averse* objective function using the dynamic risk measure within our reinforcement learning model. We also examine the reinforcement learning algorithm presented in [12] and propose improvements to enhance its performance. Finally, we outline the structure of the reinforcement learning algorithm as applied to our model.

3.1 The risk measure setting

Evaluating a model's performance based on a single path can be misleading, as it may reflect an extreme or atypical scenario. To achieve a more accurate and reliable assessment, the model should be tested multiple times. For example, instead of relying on the result from just one price path, we can simulate 100 different price paths and analyze the distribution of results across these simulations. By considering the entire distribution of outcomes, we obtain a more comprehensive understanding of the model's overall performance. However, working with distributions is more complicated than using a single value, which is why it is often preferable to summarize them with a single number that captures their key characteristics. This is where risk measures play a crucial role, particularly in finance. Risk measures allow us to quantify and evaluate the risk associated with a distribution of outcomes, simplifying the complexity into a single number that effectively captures the underlying risk. In finance, the focus is generally more on losses than profits, especially

when dealing with uncertainties. To mitigate risks effectively, we aim to improve the worst-case outcomes and address the tail of the distribution, which represents the most unfavorable results. Two commonly used risk measures for this purpose are Value at Risk (VaR)[14] and Conditional Value at Risk (CVaR)[38]. VaR estimates the maximum loss that could occur within a specific time period, while CVaR calculates the average loss when losses are under the VaR threshold.

3.1.1 VaR and CVaR

For a given confidence level $\alpha \in [0, 1]$, VaR of random variables Z is calculated as the threshold value below which α percent of the data falls. Mathematically, it is defined as:

$$\text{VaR}_\alpha(Z) = \sup\{z \in \mathbb{R} \mid \Pr(Z \leq z) \leq \alpha\}. \quad (3.1.1.1)$$

For a given confidence level $\alpha \in [0, 1]$, CVaR of random variables Z is calculated as the mean of the worst α percent of performance outcomes. It provides a measure of the expected loss given that a loss is smaller than the VaR threshold, capturing the mean of the generalized α -tail distribution. Hence it is also called Tail VaR[38]. It is expressed as:

$$\text{CVaR}_\alpha(Z) = \mathbb{E}[Z \mid Z \leq \text{VaR}_\alpha(Z)]. \quad (3.1.1.2)$$

CVaR is also often written as below:

$$\text{CVaR}_\alpha(Z) = \text{VaR}_\alpha(Z) - \frac{1}{\alpha} \mathbb{E}[(\text{VaR}_\alpha(Z) - Z)^+], \quad (3.1.1.3)$$

where $(\text{VaR}_\alpha(Z) - Z)^+$ represents the shortfall below VaR.

CVaR offers several advantages over VaR. For instance, CVaR considers the entire distribution of tail losses rather than just a single point. This allows it to reflect the severity of tail events more accurately. While VaR might ignore potential extreme losses in the tail, CVaR captures these losses, making it a more robust measure for assessing and managing tail risks. Additionally, CVaR is a coherent risk measure, whereas VaR is not [42]. A coherent risk measure satisfies four key properties: monotonicity, subadditivity, homogeneity, and translation invariance, which will be discussed later in this section. Although CVaR is similar to VaR, its coherent mathematical properties and enhanced ability to handle tail risk have made it increasingly popular in various applications. In this thesis, we will focus exclusively on CVaR risk measures and how they are used to formulate dynamic risk measures in our models.

3.1.2 Reformulation of CVaR

CVaR, as a coherent risk measure, has many useful properties. The most fundamental one is the dual representation[40]. This property allows us to reformulate CVaR in equations (3.1.1.2) and (3.1.1.3) as a form of an optimization problem. We refer to the work in [12, 44] and begin this section by presenting the necessary mathematical framework used in the reformulation.

Consider a probability space $(\Omega, \mathcal{F}, \mathbb{P}_r)$ where Ω is the sample space, \mathcal{F} is an event space and \mathbb{P}_r is a probability function. Let $p \in [1, \infty]$. The space of random variables $\mathcal{Z} = \mathcal{L}^p(\Omega, \mathcal{F}, \mathbb{P}_r)$ is defined as:

$$\mathcal{L}^p(\Omega, \mathcal{F}, \mathbb{P}_r) = \{f : \Omega \rightarrow \mathbb{R} \mid f \text{ is measurable and } \|f\|_p < \infty\},$$

with the p -norm given by:

$$\|f\|_p = \left(\int_{\Omega} |f(x)|^p d\mathbb{P}_r(x) \right)^{1/p} \quad \text{for } 1 \leq p \leq \infty.$$

Its dual space $\mathcal{Z}^* = \mathcal{L}^q(\Omega, \mathcal{F}, \mathbb{P}_r)$ is defined for $q \in [1, \infty]$ with $\frac{1}{p} + \frac{1}{q} = 1$.

For a sequence of variables $Z_t, t = 1, \dots, T$, we assume that $Z_t \in \mathcal{Z}$, which means the random variables Z_t has a finite p -th order moment with respect to the reference probability measure \mathbb{P}_r [44]. We define the coherent CVaR risk measure as $\rho : \mathcal{Z} \rightarrow \overline{\mathbb{R}}$, where $\mathcal{Z} := \mathcal{L}^p(\Omega, \mathcal{F}, \mathbb{P}_r)$. Since CVaR is a coherent risk measure [42], it possesses the following mathematical properties:

Theorem 1. *A CVaR risk measure is a mapping $\rho : \mathcal{Z} \rightarrow \overline{\mathbb{R}}$ that has the following properties for any $Z_1, Z_2 \in \mathcal{Z}$:*

- (i) **Monotonicity:** *If $Z_1 \leq Z_2$, then $\rho(Z_1) \leq \rho(Z_2)$;*
- (ii) **Translation Invariance:** *If η is a constant, $\eta \in \mathbb{R}$, then $\rho(Z_1 + \eta) = \rho(Z_1) + \eta$;*
- (iii) **Convexity:** *$\rho(\lambda Z_1 + (1 - \lambda)Z_2) \leq \lambda\rho(Z_1) + (1 - \lambda)\rho(Z_2)$ with $0 \leq \lambda \leq 1$;*
- (iv) **Positive homogeneity:** *$\rho(\lambda Z_1) = \lambda\rho(Z_1)$ with $0 \leq \lambda$;*
- (v) **Subadditivity:** *$\rho(Z_1 + Z_2) \leq \rho(Z_1) + \rho(Z_2)$;*
- (vi) **Zero case:** *$\rho(0) = 0$.*

Definition 1. A risk measure is called **proper** if it is a mapping $\rho : \mathcal{Z} \rightarrow \overline{\mathbb{R}}$ that satisfies $\rho(Z) > -\infty$ for all $Z \in \mathcal{Z}$ and has a domain defined as $\text{dom}(\rho) := \{Z \in \mathcal{Z} : \rho(Z) < +\infty\}$.

Definition 2. A coherent risk measure $\rho : \mathcal{Z} \rightarrow \overline{\mathbb{R}}$ is said to be **lower semicontinuous** if for every sequence of random variables $(Z_n) \subset \mathcal{Z}$ that converges to a random variable $Z \in \mathcal{Z}$:

$$\liminf_{n \rightarrow \infty} \rho(Z_n) \geq \rho(Z).$$

Theorem 2. Representation Theorem ([12, 44]): For a proper, convex and lower semicontinuous risk measure $\rho : \mathcal{Z} \rightarrow \overline{\mathbb{R}}$, where $\mathcal{Z} := L^p(\Omega, \mathcal{F}, \mathbb{P}_r)$ and its dual space is $\mathcal{Z}^* := L^q(\Omega, \mathcal{F}, \mathbb{P}_r)$, there exists a set $U(\mathbb{P}_r)$:

$$U(\mathbb{P}_r) = \{\xi \in B : \mathbb{E}_\xi[Z] \leq \rho(Z), \forall Z \in \mathcal{Z}\}, \quad (3.1.2.1)$$

with

$$B := \left\{ \xi \in \mathcal{Z}^* : \int_{\Omega} \xi d\mathbb{P}_r = 1, \xi \geq 0 \right\}, \quad (3.1.2.2)$$

such that

$$\rho(Z) = \sup_{\xi \in U(\mathbb{P}_r)} \mathbb{E}_\xi[Z], \quad Z \in \mathcal{Z}. \quad (3.1.2.3)$$

The expectation $\mathbb{E}_\xi[Z]$ taken with respect to the probability measure $\xi\mathbb{P}_r$ is expressed as follows:

$$\mathbb{E}_\xi[Z] = \int_{\Omega} \xi(\omega)Z(\omega)d\mathbb{P}_r(\omega). \quad (3.1.2.4)$$

A detailed proof that the CVaR risk measure is a proper, convex, and lower semicontinuous risk measure, which can be applied to Theorem 2, can be found in [44]. Consider $Z \in \mathcal{Z}$ as a cost variable. Minimizing the risk of Z , evaluated by CVaR, can be formulated as $\min \rho(Z)$. By applying Theorem 2, which links CVaR to the dual space, the cost minimization is expressed as an optimization problem, which can be written as:

$$\min_{\xi \in U(\mathbb{P}_r)} \sup_{Z \in \mathcal{Z}} \mathbb{E}_\xi[Z], \quad (3.1.2.5)$$

This reformulation facilitates the computation of risk-adjusted values by considering a distorted probability measure $\xi\mathbb{P}_r$. The reformulated CVaR measure is more adaptable in capturing varying risk preferences, as it adjusts to different probability distributions that reflect a decision-maker’s risk tolerance. In real-world decision-making, investors may not strictly follow the probabilities derived from historical data or market statistics. They might, for example, believe that the chance of extreme losses, such as market crashes or unexpected events, is higher than what the data suggests. The distorted probability measures in CVaR enable investors to reweight these scenarios based on their own beliefs, allowing for a more personalized approach to risk management. This approach also makes decision-making more robust as it accounts for worst-case scenarios by evaluating the supremum of expected outcomes under different distorted probability measures.

In financial risk management, where the future is highly interconnected with the present, decisions made at one point can have far-reaching implications on future stages. For instance, consider a trader who has measured the risk of today’s financial positions. On the second day, a trade is executed, and the market’s response unfolds. Consequently, the trader must recompute the risk of the altered position, taking into account the market’s feedback. Ignoring this future risk can lead to a suboptimal trading strategy. By evaluating risk based solely on the current stage, the current CVaR risk measure fails to consider the potential cascading effects of current timestep decisions on subsequent outcomes. Therefore, we introduce a dynamic convex risk measure methodology, which can address how risk evaluations at different times are related.

3.1.3 Dynamic convex risk measures

The core aspect of the dynamic convex risk measure is its successive measurement framework, which offers several advantages over conditional convex risk measures. A dynamic convex risk measure can incorporate the effects of both current and future states, providing a more comprehensive assessment of risk over time compared to a conditional convex risk measure. It enhances the decision-making process in long-term planning and constructs a more robust RL model. Additionally, the dynamic framework itself offers the ability to dynamically adjust to new information and evolving conditions. It is more flexible and adaptive than static conditional convex risk measures.

Another important concept mentioned in this chapter is time consistency. Time consistency means that the risk preferences or risk assessments remain stable over time. In other words, if a certain policy is deemed optimal at an initial time, it should remain optimal as time progresses, provided no new information alters the underlying risk landscape. This

property is crucial for ensuring that decisions made at different points in time are aligned and do not contradict each other.

Using a time-consistent dynamic risk measure ensures that the risk evaluations at various stages are coherent with one another, thus providing a stable and reliable framework for decision-making. This consistency helps in building trust in the decision-making process and avoids scenarios where earlier decisions are undermined by later risk assessments. Consequently, employing a time-consistent dynamic risk measure in reinforcement learning models leads to more reliable and predictable outcomes, enhancing the overall robustness and effectiveness of the model.

Consider a probability space $(\Omega, \mathcal{F}, \mathbb{P}_r)$, and a filtration $\mathcal{F}_0 \subset \mathcal{F}_1 \subset \dots \subset \mathcal{F}_T \subset \mathcal{F}$. Define the spaces $\mathcal{L}_t = \mathcal{L}^p(\Omega, \mathcal{F}_t, \mathbb{P}_r)$, $p \in [1, \infty]$, $t = 0, \dots, T$, and $\mathcal{L}_{t,T} = \mathcal{L}_t \times \mathcal{L}_{t+1} \times \dots \times \mathcal{L}_T$. Given an adapted sequence of random variables $Z = (Z_t, Z_{t+1}, \dots, Z_T)$, $Z \in \mathcal{L}_{t,T}$ holds and Z_t is a cost variable here.

Definition 3. A mapping $\rho_{t,T} : \mathcal{L}_{t,T} \rightarrow \mathcal{L}_t$ for $t = 0, \dots, T - 1$ is called a conditional risk measure if it satisfies the following monotonicity property:

$$\rho_{t,T}(Z) \leq \rho_{t,T}(W) \quad \text{for all } Z, W \in \mathcal{L}_{t,T} \text{ such that } Z \leq W. \quad (3.1.3.1)$$

Definition 4. A conditional risk measure $\rho_{t,T} : \mathcal{L}_{t,T} \rightarrow \mathcal{L}_t$ for $t = 0, \dots, T - 1$ is said to be a conditional convex risk measure if it satisfies the following properties:

- (i) **Conditional Translation Invariance:** For any $Z \in \mathcal{L}_{t,T}$ and $\eta \in \mathcal{L}_t$, $\rho(Z_1 + \eta) = \rho(Z_1) + \eta$;
- (ii) **Conditional Convexity:** For any $Z, W \in \mathcal{L}_{t,T}$ and $\lambda \in \mathcal{L}_t$ with $0 \leq \lambda \leq 1$, $\rho(\lambda Z + (1 - \lambda)W) \leq \lambda \rho(Z) + (1 - \lambda)\rho(W)$;
- (iii) **Zero case:** $\rho(0) = 0$.

Definition 5. A conditional convex risk measure $\rho_{t,T} : \mathcal{L}_{t,T} \rightarrow \mathcal{L}_t$ for $t = 0, \dots, T - 1$ is called a conditional coherent risk measure if it satisfies the positive homogeneity property.

Definition 6. A dynamic coherent risk measure is a sequence of conditional coherent risk measures $\{\rho_{t,T}\}_{t=0}^{T-1}$:

$$\rho_{t,T} : \mathcal{L}_{t,T} \rightarrow \mathcal{L}_t, \quad t = 0, \dots, T - 1. \quad (3.1.3.2)$$

Definition 7. A dynamic coherent risk measure $\{\rho_{t,T}\}_{t=0}^{T-1}$ is time consistency if the following implication holds true:

$$\rho_{t+1,T}(Z) \leq \rho_{t+1,T}(W) \implies \rho_{t,T}(0, Z) \leq \rho_{t,T}(0, W).$$

where

$$Z = (Z_{t+1}, Z_{t+2}, \dots, Z_T), W = (W_{t+1}, W_{t+2}, \dots, W_T), \quad Z, W \in \mathcal{L}_{t+1,T}$$

We have already shown the monotonicity, translation invariance, convexity, and positive homogeneity of the CVaR risk measure as established in Theorem 1. Additionally, the CVaR risk measure satisfies the zero case. According to Definition 3, 4 and 5, it is therefore a conditional coherent risk measure. By Definition 6, a sequence of CVaR risk measures can be used to construct a dynamic coherent risk measure $\rho_{t,T} : \mathcal{L}_{t,T} \rightarrow \mathcal{L}_t$, for $t = 0, \dots, T$.

However, directly implementing this dynamic coherent risk measure in its original form (3.1.3.2) poses several challenges in practice. Since the input $Z = (Z_t, Z_{t+1}, \dots, Z_T)$, with $Z \in \mathcal{L}_{t,T}$, represents a sequence of cost variables, the evaluation of the risk measure requires waiting until the entire trajectory Z is generated. Additionally, as $\rho_{t,T}$ depends on future risk measures $\rho_{t+1,T}, \dots, \rho_{T-1,T}$, when T increases, the number of possible trajectories expands exponentially, resulting in the computational cost of evaluating $\rho_{t,T}(Z)$ increasing exponentially.

Another key question for a dynamic risk measure is how risk evaluations at different times are interrelated. In dynamic settings, decisions and risk assessments made at the current stage often influence those made later. Minimizing risk at one stage might conflict with decisions required at subsequent stages, potentially leading to suboptimal or inconsistent outcomes.

To achieve consistent outcomes, it is crucial that preferences remain aligned over time. For instance, if a financial position at some future time is always preferable to another position, it should also be preferable today. Time consistency is a fundamental property of a dynamic risk measure that ensures risk assessments and decisions remain logically coherent as time progresses.

Time consistency guarantees that if a policy minimizes risk over the entire time horizon, any portion of that policy, starting from a given stage onward, also minimizes risk for the remaining time horizon[9]. This property is especially critical in multi-stage decision-making problems, ensuring that intermediate decisions do not contradict the overall risk management strategy. In essence, time consistency maintains harmony between present

and future decisions, enabling a rational and aligned approach to dynamic risk management.

Fortunately, [39] proposed a recursive reformulation to address these issues. This nested form fits naturally within the framework of dynamic programming and significantly reduces the computational burden, changing it from exponential growth to linear growth. It also allows for flexible and responsive decision-making as new information becomes available, decomposes the overall risk into step-by-step risks generated during the sequential decision-making process, and makes the risks easier to manage and interpret. Moreover, this approach ensures time consistency, maintaining coherence in risk evaluations and decisions throughout the entire decision-making process. To understand this approach, we first introduce the concept of a one-step conditional risk measure.

Definition 8. *A one-step conditional risk measure is a mapping ρ_t :*

$$\rho_t : \mathcal{L}_{t+1} \rightarrow \mathcal{L}_t, \quad t = 0, \dots, T-1. \quad (3.1.3.3)$$

such that

$$\rho_t(Z) = \rho_{t,t+1}(0, Z), \forall Z \in \mathcal{L}_{t+1}. \quad (3.1.3.4)$$

The CVaR risk measure ρ can be proved to be a one-step conditional risk measure[39]. Let ρ_t in equation (3.1.3.4) represent the CVaR risk measure for $t = 0, \dots, T-1$, Consider $\mathcal{L}_{0,T} = \mathcal{L}_0 \times \mathcal{L}_1 \times \dots \times \mathcal{L}_T$ and a sequence of cost variables $Z = (Z_0, Z_2, \dots, Z_T)$. Define $\tilde{\rho}_{0,T}$ as:

$$\tilde{\rho}_{0,T}(Z) := Z_0 + \rho_0 \left(Z_1 + \rho_1 \left(Z_2 + \dots + \rho_{T-2} (Z_{T-1} + \rho_{T-1}(Z_T)) \dots \right) \right). \quad (3.1.3.5)$$

$\tilde{\rho}_{0,T}$ is a dynamic risk measure by [41]. Notice that we may define $\tilde{\rho}_{t,T} : \mathcal{L}_{t,T} \rightarrow \mathcal{L}_t$ with $t \in [0, T]$ similarly as in (3.1.3.5):

$$\tilde{\rho}_{t,T}(Z) := Z_t + \rho_t \left(Z_{t+1} + \rho_{t+1} \left(Z_{t+2} + \dots + \rho_{T-2} (Z_{T-1} + \rho_{T-1}(Z_T)) \dots \right) \right). \quad (3.1.3.6)$$

We will refer to this as the dynamic CVaR risk measure. We will then prove the time consistency of this recursive dynamic risk measure form.

Theorem 3. Time-consistent Theorem[39]: *Suppose a dynamic risk measure $\{\rho_{t,T}\}_{t=0}^{T-1}$ satisfies the following conditions for all $Z \in \mathcal{L}$ and all $t = 0, \dots, T$:*

$$\rho_{t,T}(Z_t, Z_{t+1}, \dots, Z_T) = Z_t + \rho_{t,T}(0, Z_{t+1}, \dots, Z_T), \quad (3.1.3.7)$$

$$\rho_{t,T}(0, \dots, 0) = 0. \quad (3.1.3.8)$$

Then it is time-consistent if and only if for all $0 \leq \tau_1 \leq \tau_2 \leq T$ and all $Z \in \mathcal{L}_{0,T}$ the following identity is true:

$$\rho_{\tau_1,T}(Z_{\tau_1}, \dots, Z_{\tau_2}, \dots, Z_T) = \rho_{\tau_1,\tau_2}(Z_{\tau_1}, \dots, Z_{\tau_2-1}, \rho_{\tau_2,T}(Z_{\tau_2}, \dots, Z_T)). \quad (3.1.3.9)$$

One example presented in [39] that applies Theorem 3 is CVaR. We present this result as a theorem and provide its proof.

Theorem 4. For a dynamic CVaR risk measure $\tilde{\rho}_{t,T}$, the conditions (3.1.3.7), (3.1.3.8), and (3.1.3.9) are satisfied, and hence it is time-consistent.

Proof. Referring to Theorem 1, since for a CVaR risk measure ρ , $\rho(0) = 0$ always holds, the dynamic risk measure composed of a sequence of CVaR risk measures also satisfies the zero case, where $\tilde{\rho}_{t,T}(0, 0, \dots, 0) = 0$.

Based on the translation invariance property of CVaR, where $\rho_t(Z + W) = Z + \rho_t(W)$, $\forall Z \in \mathcal{L}_t, W \in \mathcal{L}_{t+1}$, equation (3.1.3.7) holds by default for $\tilde{\rho}_{t,T}$. Note that Z_t is assumed to be a \mathcal{F}_t -measurable random variable here.

Next, consider equation(3.1.3.6):

$$\begin{aligned} \tilde{\rho}_{\tau_1,T}(Z_{\tau_1}, \dots, Z_{\tau_2}, \dots, Z_T) &= Z_{\tau_1} + \rho_{\tau_1} \left[Z_{\tau_1+1} + \rho_{\tau_1+1} \left(Z_{\tau_1+2} + \dots \right. \right. \\ &\quad \left. \left. \dots + Z_{\tau_2} + \rho_{\tau_2} (Z_{\tau_2+1} + \dots + \rho_T(Z_T)) \dots \right) \right] \\ &= Z_{\tau_1} + \rho_{\tau_1} \left[Z_{\tau_1+1} + \rho_{\tau_1+1} \left(Z_{\tau_1+2} + \dots + \tilde{\rho}_{\tau_2,T}(Z_{\tau_2}, \dots, Z_T) \dots \right) \right] \\ &= \tilde{\rho}_{\tau_1,T}(Z_{\tau_1}, \dots, Z_{\tau_2-1}, \tilde{\rho}_{\tau_2,T}(Z_{\tau_2}, \dots, Z_T), \underbrace{0, 0, \dots, 0}_{T-\tau_2 \text{ zeros}}) \\ &= \tilde{\rho}_{\tau_1,\tau_2}(Z_{\tau_1}, \dots, Z_{\tau_2-1}, \tilde{\rho}_{\tau_2,T}(Z_{\tau_2}, \dots, Z_T)). \end{aligned}$$

Thus, equation (3.1.3.9) is satisfied. Therefore, by Theorem 3, we can conclude that the dynamic CVaR risk measure $\{\tilde{\rho}_{t,T}\}_{t=0}^{T-1}$ is time-consistent. \square

For a cost sequence c_t , $t = 0, \dots, T - 1$, we define $Z_0 = 0$ and $Z_{t+1} = c_t$ for $t = 0, \dots, T - 1$. By employing the time-consistent, dynamic convex risk measure $\{\tilde{\rho}_{t,T}\}_{t=0}^{T-1}$

formulated by CVaR ρ_t , $t \in [0, T - 1]$ in this section, we derive the following form:

$$\begin{aligned} \tilde{\rho}_{0,T-1}(Z_0, \dots, Z_{T-1}) &= \tilde{\rho}_{0,T-1}(0, c_0, \dots, c_{T-1}) \\ &= \rho_0 \left(c_0 + \rho_1 \left(c_1 + \rho_2 \left(c_2 + \dots + \rho_{T-1}(c_{T-1}) \dots \right) \right) \right). \end{aligned} \quad (3.1.3.10)$$

3.2 The RL algorithm setting

In this section, we will build an RL model to solve financial investment problems. We delve into enhancing the RL algorithm in [12]. We begin by introducing the risk-averse objective function that will be incorporated into the algorithm. The function is expressed as:

$$\min_{\pi} \tilde{\rho} \left(\sum_{t=0}^{T-1} c_t \right),$$

and will be discussed in the following. Next, we conduct a thorough examination of the original algorithm, highlighting its key features and underlying principles. Following this, we present our contributions, detailing the improvements and modifications we make.

3.2.1 Risk-averse objective function

As introduced in Section 2.2, for a *risk-neutral* reinforcement learning (RL) model with a MDP framework $\langle \mathbf{S}, \mathbf{A}, \mathbb{P}, \mathbf{R} \rangle$, the objective function is formulated as:

$$\max_{\pi} \mathbb{E} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right]. \quad (3.2.1.1)$$

However, especially in volatile environments, focusing solely on expected returns can lead to large potential losses that may outweigh the benefits of high rewards. To ensure more consistent outcomes, prioritizing stability and safeguarding against adverse scenarios, we set the goal for the RL learner as minimizing the total cost. In a *risk-averse* RL model, minimizing total risk is more important than maximizing average returns. For example, a risk-averse RL agent prefers an action with a lower but guaranteed reward over one with a potentially higher expected return but also a significant chance of loss.

In a risk-averse RL setting, we consider a MDP with the tuple $(\mathbf{S}, \mathbf{A}, \mathbb{P}, C)$, where the reward function is replaced by a cost function $C : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \rightarrow \mathbb{R}$. The cost $c_t =$

$C(s_t, a_t, s_{t+1})$ represents the negative impact incurred at time $t \in [0, T - 1]$ when the agent transitions from state s_t to state s_{t+1} via action a_t . Equation (3.2.1.1) is reformulated as:

$$\min_{\pi} \mathbb{E} \left[\sum_{t=0}^{T-1} \gamma^t c_t \right], \quad (3.2.1.2)$$

where the goal is to minimize the expected cumulative cost, which represents the objective function for a risk-averse RL model.

However, equation (3.2.1.2) only considers the expected value of the cost, ignoring the variability associated with the cost. For instance, two policies might have the same expected cost, but one could have a much higher risk of extreme negative outcomes, it is hard to tell which one is better by looking at the expected cost. Therefore, using a CVaR dynamic risk measure $\tilde{\rho}$ is a better approach to quantify and manage the cost than simply relying on expectation. Applying the risk measure to the objective function ensures that the optimal strategy is robust against potential worst-case scenarios. It helps in creating strategies that are more consistent and reliable, especially in situations where large deviations from the expected outcome could be detrimental. The new risk-averse objective function, which incorporates the CVaR dynamic risk measure $\tilde{\rho}$, is expressed as:

$$\min_{\pi} \tilde{\rho} \left(\sum_{t=0}^{T-1} \gamma^t c_t \right). \quad (3.2.1.3)$$

In portfolio optimization problems, future risks are often as important as current ones. Discounting them is not reasonable, so we set $\gamma = 1$. Hence equation (3.2.1.3) becomes:

$$\min_{\pi} \tilde{\rho} \left(\sum_{t=0}^{T-1} c_t \right). \quad (3.2.1.4)$$

As proven in Section 3.1.3, this objective function in equation (3.2.1.4) satisfies the time-consistency property, which is crucial in dynamic settings. Time consistency ensures that decisions made today remain optimal in the future as more information becomes available, thereby preventing policy shifts that could lead to suboptimal outcomes.

3.2.2 Optimization problem formulation

In this section, we extend equation (3.2.1.4) using a form of equation (3.1.3.10) to formulate an optimization problem. Consider an RL model with a modified discrete-time MDP

represented by the 4-tuple $(\mathbf{S}, \mathbf{A}, \mathbb{P}, C)$. Recall from Chapter 2 that in this RL model, a policy π is parameterized by some parameters θ . Since the policy governs the agent's decisions, different values of θ result in different actions being taken, which in turn leads to different states being reached. Therefore, in state s_t , we denote the action selected under the policy π^θ as a_t^θ , and the resulting next state as s_{t+1}^θ . The policy $\pi^\theta(a_t = a \mid s_t = s)$ represents the probability of taking action $a_t^\theta = a$ given the current state s . The term $\mathbb{P}(s_{t+1} = s' \mid s_t = s, a_t = a)$ represents the probability of transitioning from state s to state $s_{t+1}^\theta = s'$ after taking action a , based solely on the environment's dynamics. This probability is independent of the policy π^θ , meaning it does not depend on how actions are chosen by the agent, but only on the inherent behavior of the environment. The transition probability function \mathbb{P}^θ under a fixed policy π^θ is the product of the environment's inherent transition dynamics $\mathbb{P}(s_{t+1} = s' \mid s_t = s, a_t = a)$ and the policy's decision-making process $\pi^\theta(a_t = a \mid s_t = s)$, as shown below:

$$\mathbb{P}^\theta(s_{t+1} = s' \mid s_t = s, a_t = a) = \mathbb{P}(s_{t+1} = s' \mid s_t = s, a_t = a) \cdot \pi^\theta(a_t = a \mid s_t = s) \quad (3.2.2.1)$$

Consider $Z_0 = 0$ and $Z_{t+1} = c_t^\theta = C(s_t, a_t^\theta, s_{t+1}^\theta)$ for $t = 0, \dots, T-1$, where $c_t^\theta \in \mathcal{Z}_{t+1}^\theta = L^1(\{s_t\} \times \mathbf{A}_t^\theta \times \mathbf{S}_{t+1}^\theta, \mathcal{F}, \mathbb{P}^\theta)$ and $\mathcal{Z}_{t+1}^{\theta,*}$ is defined as the dual space of \mathcal{Z}_{t+1}^θ . Here, the set \mathbf{A}_t^θ represents all possible actions a_t^θ that the agent can take in the current state s_t , and \mathbf{S}_{t+1}^θ represents all possible states s_{t+1}^θ that the agent can reach after taking action a_t^θ . Recall from Theorem 2, the CVaR risk measure $\rho_t(Z_{t+1})$, $t = 0, \dots, T-1$, can be written as:

$$\rho_t(Z_{t+1}) = \sup_{\xi \in U(\mathbb{P}^\theta(s_{t+1} \mid s_t, a_t))} \mathbb{E}_\xi[Z_{t+1}], \quad t \in [0, T-1], \quad (3.2.2.2)$$

where

$$U(\mathbb{P}^\theta(s_{t+1} \mid s_t, a_t)) = \{\xi \in B : \mathbb{E}_\xi[Z_{t+1}] \leq \rho(Z_{t+1}), \forall Z_{t+1} \in \mathcal{Z}_{t+1}^\theta\}, \quad (3.2.2.3)$$

with

$$B := \left\{ \xi \in \mathcal{Z}^* : \sum_{\omega \in (\{s_t\} \times \mathbf{A}_t^\theta \times \mathbf{S}_{t+1}^\theta)} \xi(\omega) \mathbb{P}^\theta(\omega) = 1, \xi \geq 0 \right\}. \quad (3.2.2.4)$$

$\mathbb{E}_\xi[Z_{t+1}]$ is defined as:

$$\mathbb{E}_\xi[Z_{t+1}] = \sum_{(s_t, a, s')} \xi(s_t, a, s') \mathbb{P}^\theta(s' | s_t, a) Z_{t+1}(s_t, a, s'), \quad (3.2.2.5)$$

$$= \sum_{(s_t, a, s')} \xi(s_t, a, s') \mathbb{P}(s_{t+1} = s' | s_t = s_t, a_t = a,) \pi^\theta(a_t = a | s_t = s_t) Z_{t+1}(s_t, a, s'). \quad (3.2.2.6)$$

We adapt the CVaR expression shown in equation (3.2.2.2) on equation (3.1.3.10) and then refine the objective function in equation (3.2.1.4) as follows:

$$\begin{aligned} \min_{\theta} \tilde{\rho}_{0, T-1}(Z_0, \dots, Z_T) &= \min_{\theta} \tilde{\rho}_{0, T-1} \left(C(s_0, a_0^\theta, s_1^\theta), \dots, C(s_{T-1}, a_{T-1}^\theta, s_T^\theta) \right), \\ &= \min_{\theta} \tilde{\rho}_{0, T-1} \left(c_0^\theta, \dots, c_{T-1}^\theta \right), \\ &= \min_{\theta} \rho_0 \left(c_0^\theta + \rho_1 \left(c_1^\theta + \rho_2 \left(c_2^\theta + \dots + \rho_{T-1} (c_{T-1}^\theta) \dots \right) \right) \right), \\ &= \min_{\theta} \sup_{\xi \in U(\mathbb{P}^\theta(s_1 | s_0, a_0))} \left(\mathbb{E}_\xi \left(c_0^\theta + \sup_{\xi \in U(\mathbb{P}^\theta(s_2 | s_1, a_1))} \mathbb{E}_\xi \left(c_1^\theta + \dots \right. \right. \right. \\ &\quad \left. \left. \left. + \sup_{\xi \in U(\mathbb{P}^\theta(s_T | s_{T-1}, a_{T-1}))} \mathbb{E}_\xi \left(c_{T-1}^\theta \right) \dots \right) \right) \right). \end{aligned} \quad (3.2.2.7)$$

The solution θ^* for equation (3.2.2.7) will lead to the optimal policy π^{θ^*} .

To solve equation (3.2.2.7), we draw upon dynamic programming techniques and decompose the equation into a series of subproblems that can be solved sequentially. We define the value function $V_t(s)$ in a backward manner, starting from the terminal state and moving backward in time. First, consider the case when $t = T - 1$ in equation (3.2.2.7) for some θ :

$$V_{T-1}(s; \theta) = \sup_{\xi \in U(\mathbb{P}^\theta(s_T | s_{T-1} = s, a_{T-1}))} \mathbb{E}_\xi[c_{T-1}^\theta]. \quad (3.2.2.8)$$

Next, for $t = T - 2$, we can express it as:

$$V_{T-2}(s; \theta) = \sup_{\xi \in U(\mathbb{P}^\theta(s_{T-1} | s_{T-2} = s, a_{T-2}))} \mathbb{E}_\xi[c_{T-2}^\theta + V_{T-1}(s_{T-1}^\theta; \theta)]. \quad (3.2.2.9)$$

For $t \in [0, T - 3]$, we derive V_t inductively in the same backward fashion. Consequently, for $t \in [0, T - 2]$, the value function at time t is:

$$V_t(s; \theta) = \sup_{\xi \in U(\mathbb{P}^\theta(s_{t+1} | s_t = s, a_t))} \mathbb{E}_\xi[c_t^\theta + V_{t+1}(s_{t+1}^\theta; \theta)]. \quad (3.2.2.10)$$

This recursive approach defines the value function for each timestep, allowing us to break down the original optimization problem in equation (3.2.2.7) into optimizing manageable subproblems, as expressed in equations (3.2.2.8), (3.2.2.9), and (3.2.2.10).

To optimize these value functions, we employ the policy gradient approach introduced in Section 2.3. The optimal policy π^θ is learned by following the update rule $\theta \leftarrow \theta + \beta \nabla_\theta V_t$, where β is a predefined learning rate.

Theorem 5. Gradient of Value Functions[12]: $\forall s \in \mathbf{A}$, the gradient of the value function V at period $T - 1$ is given by:

$$\nabla_\theta V_{T-1}(s; \theta) = \mathbb{E}_{\xi^*} [(C(s, a_{T-1}^\theta, s_T^\theta) - \lambda^*) \nabla_\theta \log \pi^\theta(a_{T-1} = a_{T-1}^\theta \mid s_{T-1} = s)], \quad (3.2.2.11)$$

and the gradient of the value function at time $t \in [0, T - 2]$ is:

$$\begin{aligned} \nabla_\theta V_t(s; \theta) = \mathbb{E}_{\xi^*} [(C(s, a_t^\theta, s_{t+1}^\theta) + V_{t+1}(s_{t+1}^\theta; \theta) - \lambda^*) \nabla_\theta \log \pi^\theta(a_t = a_t^\theta \mid s_t = s) \\ + \nabla_\theta V_{t+1}(s_{t+1}^\theta; \theta)], \end{aligned} \quad (3.2.2.12)$$

where (ξ^*, λ^*) is any saddle-point of the Lagrangian function of equations (3.2.2.8) and (3.2.2.10) respectively.

We make the same assumptions as in [12], which states that the logarithm of the transition probabilities, $\log(\mathbb{P}^\theta)$, is differentiable with respect to θ when $\mathbb{P}^\theta \neq 0$, and that its gradient is bounded. Recall from equation (3.2.2.1) that the term $\mathbb{P}(s_{t+1} = s' \mid s_t = s, a_t = a)$ is independent of the policy π^θ . Consequently, the gradient of $\log(\mathbb{P}^\theta)$, $\nabla_\theta \log(\mathbb{P}^\theta(s_{t+1} = s' \mid s_t = s, a_t = a))$, is equal to $\nabla_\theta \log \pi^\theta(a_t = a \mid s_t = s)$.

In Theorem 5, we apply the Lagrangian method to find the optimal solution ξ^* for the optimization problem in equation (3.2.2.7). By decomposing the original problem in equation (3.2.2.7) into subproblems, as shown in equations (3.2.2.8), (3.2.2.9), and (3.2.2.10), we can compute the gradients of each subproblem provided in Theorem 5 to iteratively update θ . The saddle point is obtained when the gradients are equal to zero. The detailed proof, including the gradient derivation, Lagrangian functions, and optimality conditions, can be found in [12].

3.2.3 Improvement

In this section, we present the main contribution of our model. While we maintain the same approach for policy estimation, the novelty of our method lies in reformulating the estimation of the value function.

We have derived the formulas for calculating the value functions, as shown in equations (3.2.2.8) and (3.2.2.10), and the policy functions, as outlined in equations (3.2.2.11) and (3.2.2.12). However, in practical implementation, it is challenging to efficiently compute $\xi^{*,v}$ for V_T that maximizes

$$\mathbb{E}_\xi[c_{T-1}^\theta] = \sum_{(s_{T-1}, a, s')} \xi(s_{T-1}, a, s') \mathbb{P}(s_T = s' \mid s_{T-1} = s_{T-1}, a_{T-1} = a) \pi^\theta(a_{T-1} = a \mid s_{T-1} = s_{T-1}) C(s_{T-1}, a_{T-1}, s_T). \quad (3.2.3.1)$$

The issue for computing equation (3.2.3.1) is as follows. One common method is to perform a linear search which will involve iterating over all possible $\xi \in U$. For a discrete action space \mathbf{A} , unless \mathbf{A} is finite and has a small size, it is impractical to compute all the combinations (s_{T-1}, a, s') . In many financial applications, the allowable trade amounts are typically not limited to a small range. Consequently, iterating over all possible trade amounts a to generate all combinations of (s_{T-1}, a, s') becomes unrealistic and results in prohibitively high computational costs.

Since it is costly and impractical to compute all the combinations (s_{T-1}, a, s') needed to accurately evaluate $\mathbb{E}_\xi[c_{T-1}^\theta]$, finding the optimal $\xi^{*,v}$ that maximizes $\mathbb{E}_\xi[c_{T-1}^\theta]$ becomes challenging. Similarly, for $t = 0, \dots, T - 2$, the value function V_t faces the same issue in approximating the value of

$$\mathbb{E}_\xi[c_t^\theta + V_{t+1}(s_{t+1}^\theta; \theta)],$$

making it difficult to achieve the optimal $\xi^{*,v}$. One can approximate $\mathbb{E}_\xi[c_{T-1}^\theta]$ by generating a subset of combinations (s_{T-1}, a, s') , but the computed value will only asymptotically approach $\mathbb{E}_\xi[c_{T-1}^\theta]$ and will never exactly match the true value. Depending on different set of combinations (s_{T-1}, a, s') used, $\mathbb{E}_\xi[c_{T-1}^\theta]$ could yield different results. Furthermore, increasing the number of generated combinations significantly raises the computational cost without resolving the underlying issue.

To summarize, while theoretical solutions might exist, their practical implementation remains limited by computational resources and the intrinsic limitations of the estimation process. Another concern is that since V_t is constructed in a nested form, it incurs a higher computational cost than a linear form. The error between the practical result and the theoretical result arising in the current step also affects the next timestep, causing the deviation to accumulate like a snowball.

While finding the optimal $\xi^{*,v}$ in the value function V_t is challenging, we can utilize a specific $\xi^{*,policy}$, along with $\lambda^{*,policy}$, that forms the saddle point of the Lagrangian function of V_t , as outlined in Theorem 5. A saddle point is a point on the surface of the value function

where the gradient is zero, representing either a maximum or minimum. Since the optimal $\xi^{*,v}$ corresponds to the scenario that maximizes the value, it must lie at a saddle point. Although multiple saddle points may exist, we can narrow down the range of possible values for $\xi^{*,v}$ by focusing on the saddle points $\xi^{*,policy}$. Therefore, we propose using this $\xi^{*,policy}$ corresponding to the maximum scenario as an efficient alternative for determining the optimal $\xi^{*,v}$. This approach eliminates the need to perform an exhaustive search over all possible $\xi \in U(\mathbb{P}^\theta(s_{t+1} \mid s_t, a_t))$, which was previously required to identify $\xi^{*,v}$. As a result, it significantly reduces computational complexity by removing the necessity to compute the supremum term in V_t at each timestep. Moreover, by maintaining the same $\xi^{*,policy}$ across both value and policy functions, we ensure a tighter coupling between the two, which leads to more cohesive updates. This unified approach avoids potential conflicts between policy and value updates that could occur when separate optimal values of ξ are calculated for each function.

Previously, $\xi^{*,v}$ was computed in both the value functions, specifically in equations (3.2.2.8) and (3.2.2.10). By only calculating $\xi^{*,policy}$ in the policy functions, as shown in equations (3.2.2.11) and (3.2.2.12), and retaining it as the optimal one for the value functions, we streamline the computation, reducing both time and complexity. For simplicity, we now denote $\xi^{*,policy}$ as ξ^* . Consequently, equation (3.2.2.8) is now rewritten as:

$$V_{T-1}(s; \theta) = \mathbb{E}_{\xi^*}[c_{T-1}^\theta], \quad (3.2.3.2)$$

$$= \sum_{(s_{T-1}, a, s')} \xi^*(s_{T-1}, a, s') \mathbb{P}(s_T = s' \mid s_{T-1} = s_{T-1}, a_{T-1} = a) \pi^\theta(a_{T-1} = a \mid s_{T-1} = s_{T-1}) C(s_{T-1}, a_{T-1}, s_T). \quad (3.2.3.3)$$

Recall that we divided the objective function in equation (3.2.2.7) into a value function form in equations (3.2.2.8) and (3.2.2.10). Hence, minimizing the objective function can be treated as minimizing the value functions. For instance, to minimize $V_{T-1}(s; \theta)$, we can express the problem as:

$$\min_{\theta} V_{T-1}(s; \theta) = \min_{\theta} \mathbb{E}_{\xi^*}[c_{T-1}^\theta]. \quad (3.2.3.4)$$

As discussed earlier, it is difficult to accurately estimate $\mathbb{E}_{\xi^*}[c_{T-1}^\theta]$ since computing all the possible combinations of (s_{T-1}, a, s') is computationally infeasible. During the testing phase, the trained agent selects only the action that minimizes the expected cost at the current state s_t . Considering the performance of other suboptimal actions becomes unnecessary, as the focus should be on the performance of the best action and improving it. A greedy action $(a_t^\theta)_{\text{greedy}}$ involves selecting the action that minimizes the expected cost at

the current state s_t , as guided by the policy neural network π^θ . Specifically, for $t = T - 1$, this can be formulated as:

$$(a_{T-1}^\theta)_{\text{greedy}} = \arg \min_{a \in \mathcal{A}_{T-1}^\theta} \mathbb{E}^{\pi^\theta} [C(s_{T-1}, a, s_T)]. \quad (3.2.3.5)$$

Thus, instead of using a subset of the possible combinations of (s_{T-1}, a, s') to evaluate $\mathbb{E}_{\xi^*} [c_{T-1}^\theta]$, we focus on the specific combination that includes the greedy action. Since for any given state s_t , there is only one best action (the greedy action), this subset contains only one element $(s_{T-1}, (a_{T-1}^\theta)_{\text{greedy}}, (s_T)_{\text{greedy}})$. Therefore, the expectation operator over actions and states in the value function V_t , as seen in equations (3.2.2.8) and (3.2.2.10), is no longer necessary. This simplification aligns with the testing procedure, where the greedy action is consistently selected. The value function V_t can now be rewritten as:

$$V_{T-1}(s; \theta) = c_{T-1}^\theta = C(s_{T-1}, (a_{T-1}^\theta)_{\text{greedy}}, (s_T)_{\text{greedy}}), \quad (3.2.3.6)$$

with $(a_{T-1}^\theta)_{\text{greedy}}$ defined in equation (3.2.3.5). Similarly

$$V_t(s; \theta) = c_t^\theta = C(s_t^n, (a_t^\theta)_{\text{greedy}}, (s_{t+1}^\theta)_{\text{greedy}}) + V_{t+1}((s_{t+1}^\theta)_{\text{greedy}}; \theta), \quad (3.2.3.7)$$

where

$$(a_t^\theta)_{\text{greedy}} = \arg \min_{a \in \mathcal{A}_t^\theta} \mathbb{E}^{\pi^\theta} [C(s_t, a, s_{t+1})]. \quad (3.2.3.8)$$

By eliminating the supremum term and expectation calculation in the value functions, we can significantly reduce the computational burden and accelerate the overall algorithm. Moreover, by applying a greedy action only in the value function estimation, we ensure that exploration is concentrated on improving the policy, while exploitation is maximized during the value function estimation. This balanced approach enhances learning efficiency and can lead to better overall performance.

3.3 The RL algorithm

We present the proposed reinforcement learning algorithm here. We use the same neural network in [12] to represent both the value function V and policy, with corresponding parameters ϕ and θ . Our implementation follows an actor-critic structure, as described in Chapter 2.

The main algorithm (Algorithm 1) consists of two main steps: For each episode $l = 1, \dots, L$, the Critic Update (Algorithm 2) is followed by the Actor Update (Algorithm 3). The Critic Update focuses on updating the value function network, while the Actor Update focuses on improving the policy network. These updates are performed iteratively over a series of training loops.

We set the gradients of V^ϕ to zero at the beginning of the Critic Update algorithm. We simulate N_V trajectories of states s_t for $t = 0, \dots, T - 1$. For each state s_t , a greedy action $(a_t^\theta)_{\text{greedy}}$ is selected by π^θ . This greedy action is then executed, obtaining the next state s_{t+1} and the associated cost c_t . For each timestep $t \in [0, T - 1]$ and each trajectory $n \in [0, N_V]$, the predicted values $v_t^n = V_t^\phi(s_t^n, \theta)$ are computed. Note that the target values are calculated using our reformulated equations (3.2.3.6) and (3.2.3.7).

For $t = T - 1$, the target value is calculated as:

$$\overline{v_{T-1}^n} = V_{T-1}(s_{T-1}^n; \theta) = C(s_{T-1}^n, (a_{T-1}^\theta)_{\text{greedy}}, (s_T^\theta)_{\text{greedy}}).$$

For $t < T - 1$, the target value is calculated as:

$$\overline{v_t^n} = V_t(s_t^n; \theta) = C(s_t^n, (a_t^\theta)_{\text{greedy}}, (s_{t+1}^\theta)_{\text{greedy}}) + V_{t+1}((s_{t+1}^\theta)_{\text{greedy}}; \theta).$$

The mean squared loss between the target values and the predicted values is then calculated. An Adam optimizer step is applied to update ϕ , completing the Critic Update process.

The Actor Update algorithm shares a similar structure as the Critic one. The target values are calculated following equations (3.2.2.11) and (3.2.2.12). Since we have multiple transitions for each timestep, we take the mean of the of the gradient loss and use that as our loss here, and an Adam optimizer step is applied to update θ , completing the Actor Update process.

Algorithm 2 Main Algorithm: Actor-Critic

- 1: Initialize V network ϕ with learning rate α and policy network θ with learning rate β ;
 - 2: Initialize the environment and Adam optimizers;
 - 3: **for** each episode $l = 1, \dots, L$ **do**
 - 4: CRITIC UPDATE();
 - 5: ACTOR UPDATE();
 - 6: **end for**
 - 7: **Done**
-

Algorithm 3 Critic Update

```
1: Critic update
2: for each training loop  $i$  do
3:   Set the gradients of  $V^\phi$  to zero;
4:   Simulate  $N_V$  trajectories of  $s_t, t = 0, \dots, T - 1$ ;
5:   Select a greedy action  $(a_t^{\theta,n})_{\text{greedy}}$  for each  $n \in [1, N]$  given state  $s_t^n$ ;
6:   Execute the action  $(a_t^{\theta,n})_{\text{greedy}}$ 
7:   Observe state  $(s_{t+1}^{\theta,n})_{\text{greedy}}$  and cost  $(c_t^{\theta,n})_{\text{greedy}} = C(s_{T-1}^n, (a_{T-1}^{\theta,n})_{\text{greedy}}, (s_{t+1}^{\theta,n})_{\text{greedy}})$ ;
8:   for every timestep  $t = 0, \dots, T - 1$  do
9:     for each trajectory  $n = 1, \dots, N_V$  do
10:      Compute the predicted values  $v_t^n = V_t^\phi(s_t^n, \theta)$ ;
11:      if  $t = T - 1$  then
12:        Apply the equation (3.2.3.6) to calculate the target value;
13:          
$$\overline{v_{T-1}^n} = (c_{T-1}^{\theta,n})_{\text{greedy}};$$

14:      else
15:        Apply the equation (3.2.3.7) to calculate the target values;
16:          
$$\overline{v_t^n} = (c_t^{\theta,n})_{\text{greedy}} + V_{t+1}^\phi((s_{t+1}^{\theta,n})_{\text{greedy}}; \theta);$$

17:      end if
18:    end for
19:  Calculate the mean squared loss between the target values and the predicted values;
20:  Apply an Adam optimizer step to update  $\phi$ ;
21: Done
```

Algorithm 4 Actor Update

```
1: Actor update
2: for each training loop  $i$  do
3:   Set the gradients of  $\pi^\theta$  to zero;
4:   Simulate  $N_{\text{policy}}$  trajectories of  $s_t$ ,  $t = 0, \dots, T - 1$ ;
5:   Select a list (with size  $M_{\text{policy}}$ ) of random actions  $a_t^{\theta, n, m}$  based on  $s_t^n$  for each  $n \in [1, N_{\text{policy}}]$ ,  $m \in [1, M_{\text{policy}}]$ ;
6:   Execute action  $a_t^{\theta, n, m}$  and observe next state  $s_{t+1}^{\theta, n, m}$ , cost  $c_t^{\theta, n, m}$ , and the log probability  $\nabla_\theta \log \pi^\theta(a_t^{\theta, n, m}, s_t^n)$ ;
7:   Get a saddle-point  $(\xi^*, \lambda^*)$  and compute  $\xi_t^{*, (n, m)} = \xi^*(a_t^{\theta, n, m}, s_{t+1}^{\theta, n, m})$ ;
8:   for every timestep  $t = 0, \dots, T - 1$  do
9:     for each trajectory  $n = 1, \dots, N_{\text{policy}}$  do
10:      Compute the predicted values  $v_t^n = V_t^\phi(s_t^n, \theta)$ ;
11:      if  $t = T - 1$  then
12:        Apply the equation (3.2.3.11) to calculate the gradient  $\nabla_\theta V_t(s_t^n; \theta)$ ;
13:          
$$\text{loss}_{T-1}^{(n)} = \frac{1}{M} \sum_{m=1}^M \xi_{T-1}^{*, (n, m)} \left( c_{T-1}^{\theta, n, m} - \lambda^* \right) \nabla_\theta \log \pi^\theta(a_{T-1}^{\theta, n, m}, s_{T-1}^n);$$

14:        else
15:          Apply the equation (3.2.3.12) to calculate the gradient  $\nabla_\theta V_t(s_t^n; \theta)$ ;
16:          
$$\text{loss}_t^{(n)} = \frac{1}{M} \sum_{m=1}^M \xi_t^{*, (n, m)} \left( c_t^{\theta, n, m} + V^\phi(s_{t+1}^{\theta, n, m}; \theta) - \lambda^* \right) \nabla_\theta \log \pi^\theta(a_t^{\theta, n, m}, s_t^n);$$

17:        end if
18:      end for
19:    end for
20:    Calculate the mean of the gradient loss sequence  $\text{loss}_t^n$ ;
21:    Apply an Adam optimizer step to update  $\theta$ ;
22:  Done
```

Chapter 4

Optimal Trade Execution

One of the classic problems in the financial area is optimal trade execution[3]. It focuses on finding an optimal strategy for executing large orders within a specific time frame, aiming to maximize total expected returns while managing risk throughout the trading process. In this chapter, we will discuss how to use reinforcement learning to solve the optimal trade execution problem. Section 4.1 provides an introduction to the problem. Section 4.2 formulates the problem. Section 4.3 reviews the methodology presented in [3]. In Section 4.4, we present our contribution by constructing a reinforcement learning framework to address the problem, followed by Section 4.5, where we propose an RL algorithm based on the settings outlined in Section 4.4.

4.1 Introduction

Optimal trade execution is a well-established problem in finance. The goal of this problem is to execute a designated number of shares of a given stock within a fixed time period with the highest expected returns[8]. The execution can involve either all buying or all selling, but it is not permitted to have both buy and sell actions within a single experiment. However, focusing solely on maximizing returns can result in great uncertainty, resulting in high volatility of the returns. Obtaining a balance between return and uncertainty is important here. Hence, the optimal trade execution problem typically involves optimizing a combination of return and uncertainty nowadays[3].

Variance is a common metric used to measure the uncertainty or risk associated with returns on an investment in finance. A higher variance indicates that returns are spread

over a wider range, signifying greater uncertainty and potential volatility. Conversely, a lower variance suggests that returns are more consistent and closer to the mean, reflecting lower risk.

Instead of focusing on returns, we consider losses, which can be viewed as costs. A higher return corresponds to a smaller loss relative to the initial portfolio value, making the objective of maximizing returns equivalent to minimizing losses. In [3], the shortfall x is defined as the loss, calculated as the difference between the initial portfolio value and the value of the portfolio at the end.

Optimizing a combination of return and uncertainty can thus be reframed as optimizing a combination of the expectation of shortfall x and its variance. The optimal trade execution problem can be formulated as the following optimization problem:

$$\min_x (E(x) + \lambda \text{Var}(x)), \quad (4.1.1)$$

where $\lambda \in [0, \infty)$ represents the weight given to variance. A higher λ indicates greater emphasis on variance. If $\lambda = 0$, the problem reduces to a standard wealth-maximization (shortfall-minimization) scenario, where maximizing returns is the only concern. In [3], the researchers developed a discrete-time, stochastic control framework to solve equation (4.1.1). This method is called as Almgren-Chriss model and will be introduced in Section 4.3.

4.2 Problem Formulation

We formulate the optimal trade execution problem following the definition in [3], focusing exclusively on the all selling scenario. For a trading horizon T , we divide it into N intervals, resulting in a total of $N + 1$ timestep points. The length of time between two consecutive timesteps is denoted as $\tau = \frac{T}{N}$.

Initially, at the discrete-time $t = 0$, the agent holds Q shares of a stock. The task is to determine the amount u_t of the shares to sell at each timestep t for $t \in [1, N]$. By the end of the trading period, at $t = N$, all shares must be sold, leaving a zero inventory. The inventory size throughout the entire trading trajectory can be defined as a sequence q_t , $t = 0, \dots, N$, where:

$$q_t = Q - \sum_{j=1}^t u_j = \sum_{j=t+1}^N u_j, \quad t = 1, \dots, N, \quad q_0 = Q = \sum_{j=1}^N u_j \quad \text{and} \quad q_N = 0. \quad (4.2.1)$$

The selling amount u_t can also be written as $u_t = q_{t-1} - q_t$, for $t = 1, \dots, N$.

When attempting to sell a fixed number of shares, it is highly likely that the highest bidding price will not be able to take the entire quantity. Consequently, some shares may be sold at lower prices than initially expected. In finance, this phenomenon is known as slippage[18], where the slippage rate quantifies the difference between the expected and actual transaction prices.

Therefore, when shares are sold at a time step t with a corresponding price X_t , the actual average transaction price is typically lower than X_t . To model this situation, we use a temporary impact function h , where the input to h at time step t is the current selling rate, expressed as $\frac{u_t}{\tau}$. In [3], the temporary impact function $h\left(\frac{u_t}{\tau}\right)$ is defined as:

$$h\left(\frac{u_t}{\tau}\right) = \epsilon \operatorname{sgn}(u_t) + \eta \frac{u_t}{\tau}, \quad (4.2.2)$$

where sgn is the sign function and ϵ is used as fixed costs of selling. When we perform a sell action u_t with a price X_{t-1} from the previous timestep, the average selling price X_t^{average} at timestep t is given by:

$$X_t^{\text{average}} = X_{t-1} - h\left(\frac{u_t}{\tau}\right), t \in [1, N], \quad (4.2.3)$$

where $h\left(\frac{u_t}{\tau}\right)$ is defined in equation (4.2.2).

Besides the temporary impact on the market price, there also exists a permanent impact. Permanent impact refers to changes in the “equilibrium” price[3]. Every time we perform a trade, especially when the trade amount is large, we adjust the balance between supply and demand in the market. The permanent price impact function is denoted as g , where the input of the function g at time step t is $\frac{u_t}{\tau}$, the current selling rate. A linear permanent impact function $g\left(\frac{u_t}{\tau}\right)$ in [3] is defined as:

$$g\left(\frac{u_t}{\tau}\right) = \zeta \frac{u_t}{\tau}, \quad (4.2.4)$$

where the constant ζ is the permanent price impact factor.

The price of a stock is influenced by three main factors: drift rate, volatility, and market impact. Drift rate and volatility arise from the stochastic processes that drive the stock’s price, while market impact, which is divided into temporary and permanent price impacts, reflects the market’s response to trading actions. Temporary price impact is short-lived, affecting only the trade price at the current timestep, whereas permanent price impact influences future prices. Therefore, we focus exclusively on the permanent price impact

here. The price evolution model in [3] assumes that the agent has no knowledge of future price movements, and thus, does not include a drift term. The price evolution is described by the following equation:

$$X_t = X_{t-1} + \sigma\sqrt{\tau}\xi_t - \tau g\left(\frac{u_t}{\tau}\right), \quad t \in [1, N], \quad (4.2.5)$$

where σ represents the volatility of the asset, ξ_t are independent draws from a standard normal distribution.

With a given initial stock price X_0 , the initial portfolio value of our position is calculated as X_0Q . The cumulative money earned at each timestep presents the total wealth at the end, shown as:

$$\sum_{t=1}^N u_t X_t^{\text{average}} = X_0Q + \sum_{t=1}^N \left(\sigma\sqrt{\tau}\xi_t - \tau g\left(\frac{u_t}{\tau}\right) \right) q_t - \sum_{t=1}^N u_t h\left(\frac{u_t}{\tau}\right). \quad (4.2.6)$$

A detailed proof of the validity of equation (4.2.6) can be found in [3]. Mathematically, we denote the shortfall x as:

$$\begin{aligned} x &= X_0Q - \sum_{t=1}^N u_t X_t^{\text{average}}, \\ &= - \sum_{t=1}^N \left(\sigma\sqrt{\tau}\xi_t - \tau g\left(\frac{u_t}{\tau}\right) \right) q_t + \sum_{t=1}^N u_t h\left(\frac{u_t}{\tau}\right). \end{aligned} \quad (4.2.7)$$

Recall from equation (4.1.1), we denote $E(x)$ for the expected shortfall and $Var(x)$ for the variance of the shortfall. The λ parameter here is a measure of risk aversion. The higher the value of λ , the more we are willing to prioritize reducing variance, even if it means sacrificing some expected cost. Overall, the mean-variance optimization problem is represented as:

$$\min_x (E(x) + \lambda Var(x)), \quad (4.2.8)$$

with x defined in equation (4.2.7).

4.3 Almgren-Chriss Model

The Almgren-Chriss model [3] employs a discrete-time, stochastic control framework to derive the optimal trade list u_t , for $t \in [0, N-1]$. In this section, we outline the construction of this framework and present the model's solution for deriving the optimal trades.

With x defined in equation(4.2.7), we readily compute $E(x)$ and $Var(x)$ as follows:

$$E(x) = \sum_{t=1}^N \tau q_t g\left(\frac{u_t}{\tau}\right) + \sum_{t=1}^N u_t h\left(\frac{u_t}{\tau}\right), \quad (4.3.1)$$

$$Var(x) = \sigma^2 \sum_{t=1}^N \tau q_t^2. \quad (4.3.2)$$

Substituting $g\left(\frac{u_t}{\tau}\right)$ and $h\left(\frac{u_t}{\tau}\right)$ in equations (4.2.2) and (4.2.4) into (4.3.1), we obtain

$$\begin{aligned} E(x) &= \sum_{t=1}^N \tau q_t \zeta \frac{u_t}{\tau} + \sum_{t=1}^N u_t \left(\epsilon \operatorname{sgn}(u_t) + \eta \frac{u_t}{\tau} \right), \\ &= \sum_{t=1}^N q_t \zeta u_t + \sum_{t=1}^N u_t \epsilon \operatorname{sgn}(u_t) + \sum_{t=1}^N \eta \frac{u_t^2}{\tau}, \\ &= \zeta \sum_{t=1}^N q_t u_t + \epsilon \sum_{t=1}^N |u_t| + \eta \sum_{t=1}^N \frac{u_t^2}{\tau}, \\ &= \zeta \sum_{t=1}^N q_t (q_{t-1} - q_t) + \epsilon \sum_{t=1}^N |u_t| + \eta \sum_{t=1}^N \frac{u_t^2}{\tau}, \\ &= \zeta \sum_{t=1}^N (q_t q_{t-1} - q_t^2) + \epsilon \sum_{t=1}^N |u_t| + \eta \sum_{t=1}^N \frac{u_t^2}{\tau}, \\ &= \frac{1}{2} \zeta \sum_{t=1}^N (q_{t-1}^2 - q_t^2 - (q_t - q_{t-1})^2) + \epsilon \sum_{t=1}^N |u_t| + \eta \sum_{t=1}^N \frac{u_t^2}{\tau}, \\ &= \frac{1}{2} \zeta Q^2 - \frac{1}{2} \zeta \sum_{t=1}^N u_t^2 + \epsilon \sum_{t=1}^N |u_t| + \eta \sum_{t=1}^N \frac{u_t^2}{\tau}, \\ &= \frac{1}{2} \zeta Q^2 + \epsilon \sum_{t=1}^N |u_t| + \left(\frac{\eta}{\tau} - \frac{1}{2} \zeta \right) \sum_{t=1}^N u_t^2, \\ &= \frac{1}{2} \zeta Q^2 + \epsilon \sum_{t=1}^N |u_t| + \frac{\tilde{\eta}}{\tau} \sum_{t=1}^N u_t^2, \end{aligned} \quad (4.3.3)$$

where $\tilde{\eta} = (\eta - \frac{1}{2}\zeta\tau)$. Note here we only consider a pure sell case, therefore $\sum_{t=1}^N |u_t| =$

$|Q| = Q$. With respect to q_1, \dots, q_N , we can rewrite equation (4.3.3) as

$$E(x) = \frac{1}{2}\zeta Q^2 + \epsilon Q + \frac{\tilde{\eta}}{\tau} \sum_{t=1}^N (q_{t-1} - q_t)^2. \quad (4.3.4)$$

Consider (4.3.2) and (4.3.4), $J(x) = E(x) + \lambda Var(x)$ can be seen as a quadratic function with q_1, \dots, q_N ; and it is strictly convex for $\lambda \geq 0$ [3]. Hence, we can obtain the unique global minimum by setting its partial derivatives to zero. More precisely, the derivative w.r.t. q_t , $t \in [1, N - 1]$ is given by:

$$\frac{\partial J}{\partial q_t} = 2\tau \left(\lambda\sigma^2 q_t - \frac{\tilde{\eta}}{\tau^2} (q_{t-1} - 2q_t + q_{t+1}) \right). \quad (4.3.5)$$

Setting equation (4.3.5) to zero, we have:

$$\frac{1}{\tau^2} (q_{j-1} - 2q_j + q_{j+1}) = \tilde{\kappa}^2 q_j, \quad (4.3.6)$$

with

$$\tilde{\kappa}^2 = \frac{\lambda\sigma^2}{\tilde{\eta}} = \frac{\lambda\sigma^2}{(\eta - \frac{1}{2}\zeta\tau)} = \frac{\lambda\sigma^2}{\eta \left(1 - \frac{\zeta\tau}{2\eta}\right)}. \quad (4.3.7)$$

The final solution of equation (4.3.7) is a combination of the exponential $\exp(\pm\kappa t\tau)$, where κ satisfies

$$\frac{2}{\tau^2} (\cosh(\kappa\tau) - 1) = \tilde{\kappa}^2. \quad (4.3.8)$$

Solving equation (4.3.6) with $\tilde{\kappa}$ satisfies equation (4.3.8), we obtain the optimal inventory list $\{q_t\}_{t=0}^N$ and its corresponding trade list $\{u_t\}_{t=1}^N$. Performing a trade amount u_t at each timestep $t \in [1, N]$ make us able to achieve the minimum value of $J(x) = E(x) + \lambda Var(x)$. The detailed proof and calculation procedures can be found in [21]. The form of $\{q_t\}_{t=0}^N$ and $\{u_t\}_{t=1}^N$ are shown as:

$$q_t = \frac{\sinh(\kappa(T - t\tau))}{\sinh(\kappa T)} Q, \quad t = 0, \dots, N - 1, \quad q_N = 0, \quad (4.3.9)$$

and the corresponding trade list is calculated by:

$$u_t = \frac{2 \sinh\left(\frac{1}{2}\kappa\tau\right)}{\sinh(\kappa T)} \cosh\left(\kappa\left(T - \left(t - \frac{1}{2}\right)\tau\right)\right) Q, \quad t = 1, \dots, N, \quad (4.3.10)$$

This approach is efficient when the price impact functions and price evolution are modeled linearly. However, in the case of nonlinear price impact functions, the objective function $J(x) = E(x) + \lambda Var(x)$ can become non-convex, potentially resulting in multiple local minima. In such scenarios, finding the global optimal solution using partial derivative methods is difficult. Moreover, with nonlinear models, it is typically only possible to obtain an approximate solution rather than a closed-form solution, which is an explicit mathematical expression that can be evaluated in a finite number of operations. In real-world scenarios, price impact functions are often far more complex and nonlinear, making it challenging to apply the same approach to address the mean-variance problem.

4.4 The RL setting

In this section, we contribute an RL framework to find the optimal trade list for the mean-variance problem. There are a few advantages to using an RL model to solve the optimal trade execution problem. The RL approach can easily adjust to adapt to dynamic market conditions. It handles high-dimensional state spaces and non-linear price impact conditions effectively. It is well-suited for complex financial tasks where traditional methods, such as PDEs, may fall short.

Recall from Section 2.2, the MDP used here is a 4-tuple $(\mathbf{S}, \mathbf{A}, \mathbb{P}, \mathbf{R})$, where each element will be defined and described in the following paragraphs.

4.4.1 Action

An action $u_t \in \mathbf{A}$, where $\mathbf{A} = [0, 1]$, determines the percentage of the current inventory of shares that needs to be sold:

- $u_t = 0$ indicates that no shares are sold at time t ;
- $u_t = 1$ indicates that the entire current inventory is sold at time t ;
- For $u_t \in (0, 1)$, u_t represents the proportion of shares on hand to be sold at time t .

4.4.2 State

The state space \mathbf{S} in this model is a 3-dimensional space, where each state $s_t \in \mathbf{S}$ represents the current state at time t . It is composed of the following three elements:

- **Current price** $X_t \in \mathbb{R}$: The price of the asset at time t , which is a real number representing the current market price.
- **Inventory left** $q_t \in [0, Q]$: The remaining shares on hand at time t , where Q is the initial inventory.
- **Timestep** $t \in [0, N]$: The current timestep t , where N is the total number of timesteps in the trading horizon.

Note that when selling a large amount of shares, even a small update in the policy can result in a significant change in u_t . This can cause substantial variations in the value of q_t , leading to large fluctuations and instability in the training process. To accelerate the convergence of the RL algorithm and prevent the feature q_t from dominating the learning process, we rescale it by dividing the inventory holding q_t at each timestep by the average inventory factor. This factor is defined as $q_{\text{average}} = \frac{Q}{N}$. Consequently, the range for q_t changes from $[0, Q]$ to $[0, N]$. Thus, the state space $\mathbf{S} = \mathbb{R} \times [0, N] \times [0, N]$ and a state $s_t = (X_t, q_t, t)$.

4.4.3 Transition Probability

There is nothing special about the transition probability function \mathbb{P} used here. It is still assumed stationary and unknown to the agent. $\mathbb{P}(s_{t+1} = s' \mid s_t = s, a_t = a)$ denotes the probability of arriving the next state s' after an action a is executed in the current state s at the time t .

4.4.4 Reward

Reward shaping is a critical component in our RL model. It is where our main contribution lies. In typical scenarios, the objective is simply to minimize costs. However, in the optimal trade execution problem, the objective is to minimize both the expected cost and the variance of the cost, as shown by the expression:

$$\min (E(x) + \lambda Var(x)),$$

where x represents the total cost (shortfall). While the expected cost $E(x)$ can be computed incrementally at each timestep, variance is more complex—it can only be calculated at the final timestep, as it requires information from the entire process.

In our RL model, at timestep t , the agent selects an action u_t in a given state s_t . The number of shares sold is calculated as $u_t \cdot q_{\text{average}}$. The money earned at this timestep is determined by the number of shares sold, multiplied by the average selling price as shown in equation (4.2.3). The expected selling money is calculated by multiplying the number of shares sold with the initial price X_0 . The difference between the expected value and the actual money earned is defined as the cost for this timestep. We define a cost function C that determines the cost at timestep t as follows:

$$c_t = C(s_t, a_t, s_{t+1}) = u_t \cdot q_{\text{average}} \cdot (X_0 - X_t^{\text{average}}). \quad (4.4.4.1)$$

The shortfall value x at the end can be expressed as the cumulative cost, i.e., the sum of c_t from $t = 0$ to $T - 1$, which represents the difference between the initial portfolio value and the actual proceeds obtained from selling over the trajectory:

$$x = \sum_{t=0}^{N-1} c_t. \quad (4.4.4.2)$$

Note that if the agent has sold all the shares at timestep k , then for $t = k + 1, \dots, N - 1$, we have $c_t = 0$.

Variance measures the spread of a data distribution by comparing each point to the mean, which requires multiple data points. With only one data point, there is no information about the spread. Since a single trajectory yields just one shortfall value, it is insufficient to compute the variance. Therefore, we generate multiple price paths (denoted as W) to calculate the variance. Note that for a given state s_t^w , where $t \in [0, N - 1]$ and $w \in [0, W]$, calculating the variance based on the current cumulative cost $\sum_{k=0}^t c_k^w$ at timestep t is not an accurate approach for computing the mean-variance value. This is because the input for the variance, which is the shortfall x , should be the total cost incurred once all shares have been sold. As long as there are remaining shares at timestep t , the current cumulative cost $\sum_{k=0}^t c_k^w$ does not equate to the shortfall x of this price path and thus cannot be used to compute the mean-variance value.

To address this, we will assume the agent is forced to sell all remaining shares at the next timestep ($t + 1$), ensuring a complete and finished trade list. This assumption allows us to calculate the shortfall x_{t+1}^w as the cumulative cost $\sum_{k=0}^{t+1} c_k^w = \sum_{k=0}^{N-1} c_k^w = x_{t+1}^w$. Hence we take x_{t+1}^w as input to calculate the mean-variance value j_t^w at time t :

$$j_t^w = J(x_{t+1}^w) = E(x_{t+1}^w) + \lambda \text{Var}(x_{t+1}^w) \quad (4.4.4.3)$$

Note that if $t = N - 1$, we can directly calculate the j_t^w value, as all remaining inventory is sold at this timestep. If at a specific $t^{*,w} \in [0, N - 1]$, the agent has already sold everything

with $q_{t^*,w+1}^w = 0$, we assign $j_k^w = j_{t^*}^w$ for $k \in [t^*,w + 1, N - 1]$. We refer to this specific t^*,w as the finished index.

The primary goal of the RL agent is to find an optimal trade list that minimizes the mean-variance value. We can encourage the action that leads to a smaller value of j_t^w by setting $-j_t^w$ as the reward r_t^w . In this case, the agent would maximize r_t^w , which is equivalent to minimizing j_t^w . However, this approach raises some issues. Since the ultimate objective is to minimize the mean-variance value at the final timestep, j_{N-1}^w , focusing on minimizing j_t^w at each individual timestep could lead to suboptimal results. An action may increase j_t^w temporarily but result in a better next state s_{t+1}^w , where the agent can achieve a lower j_{t+1}^w by selecting a more effective action, ultimately leading to a lower j_{N-1}^w . Additionally, because this is a pure selling case with no buying allowed, the inventory q_t^w is always non-increasing over time. When the agent is forced to sell all remaining shares, a larger inventory q_t^w will lead to a higher expected shortfall, even if the variance improves slightly, ultimately resulting in a higher value of j_t^w . Hence, j_t^w at later timesteps is likely to have a higher value than j_t^w at earlier timesteps, and this discrepancy is not due to better actions taken at later timesteps, but rather because q_t^w is typically smaller as time progresses. Not only does the selling amount u_t^w contribute to improving j_t^w , but all previous actions u_i^w for $i = 0, \dots, t - 1$ also play a role in lowering j_t^w , as these actions collectively result in a reduced q_t^w . Thus, j_t^w is not an appropriate choice for the immediate reward r_t^w . To define a more suitable reward for use here, we first need to store a baseline inventory list within our model.

Therefore, j_t^w is not a suitable choice for the immediate reward r_t^w . To find a better way for defining the reward, we first need to store a baseline inventory list in our model. At the beginning of the algorithm, we initialize it with W random trajectories. For each state $s_t^{\text{baseline},w}$, the agent performs an action $u_t^{(\text{baseline},w)}$, resulting in the next timestep state $s_{t+1}^{\text{baseline},w}$. We calculate $x_{t+1}^{\text{baseline},w}$ for each state $s_t^{\text{baseline},w}$, and then compute $j_t^{\text{baseline},w}$ according to equation (4.4.4.3). The mean-variance value is then obtained by taking the average over W trajectories, shown as:

$$j_t^{\text{baseline}} = \frac{\sum_{w=1}^W j_t^{(\text{baseline},w)}}{W}. \quad (4.4.4.4)$$

Similarly, we compute the baseline trade list as

$$u_t^{\text{baseline}} = \frac{\sum_{w=1}^W u_t^{(\text{baseline},w)}}{W}. \quad (4.4.4.5)$$

We calculate the current average mean-variance value j_t as the mean of j_t^w across all

trajectories, defined as:

$$j_t = \frac{\sum_{w=1}^W j_t^w}{W}, \text{ for } t \in [0, N - 1]. \quad (4.4.4.6)$$

Next, we compare j_{N-1} with $j_{N-1}^{\text{baseline}}$. If j_{N-1} is smaller than $j_{N-1}^{\text{baseline}}$, we replace the baseline list $\{j_t^{\text{baseline}}\}_{t=0}^{N-1}$ with the current mean-variance list $\{j_t\}_{t=0}^{N-1}$. Following this, the baseline trade list $\{u_t^{\text{baseline}}\}_{t=0}^{N-1}$ is recalculated using equation (4.4.4.5), based on the current trade list $\{u_t^w\}_{t=0}^{N-1}$. Hence, this baseline list stores the results of the trajectories that have achieved the minimum j_{N-1} value up to this point.

We then redefine the reward as $r_t^w = (j_t^{\text{baseline}} - j_t^w)$. This reward formulation helps mitigate the issue where j_t^w values are typically larger at earlier timesteps and smaller at later timesteps, which is not necessarily due to poor actions at the start but rather due to the nature of inventory reduction over time. By subtracting the corresponding baseline j_t^{baseline} , the reward reflects relative improvement, ensuring that the agent focuses on outperforming the baseline, regardless of whether j_t^w is smaller or larger in absolute terms. Additionally, this approach promotes consistency in evaluating actions across timesteps, rewarding the agent based on its ability to exceed expectations rather than simply favoring later timesteps.

Recall that for a large Q , both the expected value and variance are high, resulting in a larger j_t^w . Since we rescale Q , we must also rescale the reward accordingly. This is done by dividing the reward by j_t^{baseline} , leading to the following adjusted reward:

$$\overline{r}_t^w = \frac{(j_t^{\text{baseline}} - j_t^w)}{j_t^{\text{baseline}}}. \quad (4.4.4.7)$$

This scaling ensures that the rewards are normalized to a much smaller range, avoiding large fluctuations caused by the original scale. The primary advantage of this scaled reward is that it prevents the model from being overly sensitive to large changes in Q , resulting in more stable learning across different scenarios.

Additionally, we introduce a bonus to the reward if j_{N-1} is smaller than $j_{N-1}^{\text{baseline}}$, which indicates that the current mean-variance value at the end is lower than the previous historical minimum. Specifically, the reward at the finish index timestep $t^{*,w}$ for each path will receive a larger boost of 10 and for each timestep $t \in [0, t^{*,w}]$, the reward r_t will be increased by 1. Since the RL model is designed to maximize cumulative rewards, we aim to highlight that while intermediate actions are important, minimizing the final outcome—represented by j_{N-1} —is the ultimate objective. This reward bonus compensates and incentivizes the agent to make decisions in earlier steps that may lead to a slightly lower immediate reward but result in a smaller overall j_{N-1} .

The choice of adding 1 to the intermediate timesteps and 10 to the finish index timestep is intentional. The value of 1 is chosen because the rewards are normalized, offering a balanced incentive without dominating the learning process. Meanwhile, the value of 10 for the finish index timestep emphasizes the significance of the final mean-variance outcome, while not being excessively high. The number 10 is a balanced choice, but one could also opt for values like 15 or 20 for the finish index reward to further stress the importance of minimizing j_{N-1} . Note that this bonus will not apply if j_{N-1} is equal to or greater than $j_{N-1}^{\text{baseline}}$. Finally, we can define our reward as:

$$r_t^w = \begin{cases} \frac{(j_t^{\text{baseline}} - j_t^w)}{j_t^{\text{baseline}}} + 1, & \text{if } j_{N-1} < j_{N-1}^{\text{baseline}} \text{ and } t < t^* \\ \frac{(j_t^{\text{baseline}} - j_t^w)}{j_t^{\text{baseline}}} + 10, & \text{if } j_{N-1} < j_{N-1}^{\text{baseline}} \text{ and } t = t^* \\ \frac{(j_t^{\text{baseline}} - j_t^w)}{j_t^{\text{baseline}}}, & \text{if } j_{N-1} \geq j_{N-1}^{\text{baseline}} \text{ and } t \leq t^* \\ 0, & \text{if } t > t^{*,w} \end{cases} \quad (4.4.4.8)$$

where $t \in [0, N - 1]$.

Overall, the proposed reward shaping method in our RL model is able to evaluate both loss and variance effects at each timestep. By incorporating a rescaled reward based on a baseline, the model ensures fair comparisons across timesteps and accurately reflects improvements in the mean-variance value. This approach encourages early sacrifices for long-term gains and prevents the agent from getting stuck in local minima. The combination of comparing the baseline mean-variance value and providing extra reward bonuses for achieving a smaller $J(x)$ value offers a clear guide for the agent to learn and converge toward an optimal trading strategy, accelerating the convergence of the learning process.

4.4.5 Counter

Since the optimal trade list is unknown, the agent cannot determine whether the RL model has already found the optimal trade list or not. As a result, the agent does not know when to stop training. Typically, a fixed number of training loops l is set for the entire RL algorithm, and the final result after l loops is treated as the optimal trade list. However, this approach is inefficient because, even if the optimal solution is found early on, the model continues training for all l loops, wasting a lot of time.

To improve the efficiency of the RL training process, we introduce a counter to track updates to the baseline trade list, which represents the trade list with the smallest mean-variance value found so far. The counter is initialized to 0 and increments by 1 with each

training loop of the value network and policy network. Whenever a trade list with a smaller mean-variance value is found, the baseline trade list is replaced by this new trade list, and the counter is reset to 0. The counter thus tracks how many loops have passed since the last update to the baseline trade list.

If the baseline trade list remains unchanged for a significant number of loops, it is likely that the current baseline trade list is the optimal one. In this case, if the counter value exceeds a fixed threshold l_{stop} , we terminate the training and present the current baseline trade list as the optimal solution.

4.5 The RL algorithm

In this section, we introduce the proposed reinforcement learning algorithm, following the setting in Section 3.3. A fully-connected, multi-layer feedforward artificial neural network (ANN) is employed to model both the value function V and the policy function π , with parameters ϕ and θ , respectively. The algorithm adopts a standard actor-critic structure, utilizing the Bellman equation, Temporal Difference (TD) learning, and the Policy Gradient Method, as discussed in Chapter 2, to update the policy network π (parameterized by θ) and the value network V (parameterized by ϕ).

The main algorithm (Algorithm 4) consists of two primary steps: for each episode $l = 1, \dots, L$, the Critic Update (Algorithm 5) is followed by the Actor Update (Algorithm 6). The Critic Update focuses on updating the value function network, while the Actor Update focuses on improving the policy network. These updates are performed iteratively over a series of training loops.

As the number of trajectories used in the V function and the policy function may differ, we denote them by W_V and W_{policy} , respectively. At the beginning of the Critic Update algorithm, we first check whether the counter is bigger than l_{stop} . If yes then we stop the whole training process. We then set the gradients of V^ϕ to zero. We simulate W_V trajectories of states s_t for $t = 0, \dots, N - 1$. For each state s_t , the next state s_{t+1} is obtained using the action u_t^θ . We calculate the associated mean-variance value j_t . For each timestep $t \in [0, N - 1]$ and each trajectory $w \in [0, W_V]$, the TD error δ_t for $t \in [0, N - 1]$ is computed at each timestep t . We update ϕ according to equation (4.4.5.3).

The Actor Update algorithm shares a similar structure to the Critic Update. We check the counter. We simulate W_{policy} trajectories of states s_t for $t = 0, \dots, N - 1$. We calculate the mean-variance value and TD error used here. We update θ following equation (4.4.5.4). The details can be viewed in Algorithm 6 below.

At the end of both the Actor Update and Critic Update, we compare j_{N-1} and $j_{N-1}^{\text{baseline}}$. If j_{N-1} is smaller, we update the baseline trade list and reset the counter to 0. Otherwise, the counter is incremented by one.

Algorithm 5 Main Algorithm: Actor-Critic

- 1: Initialize critic network $V_\phi(s)$ with learning rate α and policy network π_θ with learning rate β ;
 - 2: Initialize the environment and Adam optimizers;
 - 3: Initialize the counter and baseline trade list with finished index number, obtain the baseline mean-variance list $\{j_t^{\text{baseline}}\}_{t=0}^{N-1}$;
 - 4: **for** each episode $l = 1, \dots, L$ **do**
 - 5: CRITIC UPDATE();
 - 6: ACTOR UPDATE();
 - 7: **end for**
 - 8: **Done**
-

Algorithm 6 Critic Update

```
1: Critic update
2: for each training loop  $i$  do
3:   if  $counter > l_{\text{stop}}$  then
4:     stop training process;
5:   end if
6:   Set the gradients of  $V^\phi$  to zero;
7:   Simulate  $W_V$  trajectories of  $s_t, t = 0, \dots, N - 1$ ;
8:   Obtain from  $\pi_\theta$  the associated actions  $u_t^w$  and the next timestep state  $s_{t+1}^w$  for
    $w = 1, \dots, W_V$ ;
9:   Compute the mean-variance value  $j_t^w$  according to equation (4.4.4.3) and the reward
    $r_t^w$  according to equation (4.4.4.8) for  $t \in [0, N - 1]$ ;
10:  Compute the average mean-variance value  $j_t$  according to equation (4.4.4.6) for
    $t \in [0, N - 1]$ ;
11:  for every timestep  $t = 0, \dots, N - 1$  do
12:    for each trajectory  $w = 1, \dots, W_V$  do
13:      Compute TD error:  $\delta_t = r_t^w + \gamma V_\phi(s_{t+1}^w) - V_\phi(s_t^w)$ ;
14:    end for
15:  end for
16:  if  $j_{N-1} > j_{N-1}^{\text{baseline}}$  then
17:     $j_t^{\text{baseline}} = j_t$  for  $t = 0, \dots, N - 1$ ;
18:    counter = 0;
19:  else
20:    counter = counter + 1;
21:  end if
22: end for
23: Update  $\phi$  as follows:  $\phi \leftarrow \phi + \alpha \delta_t \nabla_\phi V_\phi(s_t)$  using an adam optimizer step;
24: Done
```

Algorithm 7 Actor Update

```
1: Actor update
2: for each training loop  $i$  do
3:   if  $counter > l_{\text{stop}}$  then
4:     stop training process;
5:   end if
6:   Set the gradients of  $\pi^\theta$  to zero;
7:   Simulate  $W_{\text{policy}}$  trajectories of  $s_t, t = 0, \dots, N - 1$ ;
8:   Obtain from  $\pi_\theta$  the associated actions  $u_t^w$  and the next timestep state  $s_{t+1}^w$  for
    $w = 1, \dots, W_{\text{policy}}$ ;
9:   Compute the mean-variance value  $j_t^w$  according to equation (4.4.4.3) and the reward
    $r_t^w$  according to equation (4.4.4.8) for  $t \in [0, N - 1]$ ;
10:  Compute the average mean-variance value  $j_t$  according to equation (4.4.4.6) for
    $t \in [0, N - 1]$ ;
11:  for every timestep  $t = 0, \dots, N - 1$  do
12:    for each trajectory  $w = 1, \dots, W_{\text{policy}}$  do
13:      Compute TD error:  $\delta_t = r_t^w + \gamma V_\phi(s_{t+1}^w) - V_\phi(s_t^w)$ ;
14:    end for
15:  end for
16:  if  $j_{N-1} > j_{N-1}^{\text{baseline}}$  then
17:     $j_t^{\text{baseline}} = j_t$  for  $t = 0, \dots, N - 1$ ;
18:    counter = 0;
19:  else
20:    counter = counter + 1;
21:  end if
22: end for
23: Update  $\theta$  as follows:  $\theta \leftarrow \theta + \beta \nabla_\theta \log \pi_\theta(u_t, s_t) \delta_t$  using an adam optimizer step;
24: Done
```

Chapter 5

Experiment

In this chapter, we present the numerical results of two cases. The first case in section 5.1 relates to the dynamic CVaR risk measures discussed in Chapter 3, while the second case in section 5.2 involves the experimental results for optimal trade execution, as covered in Chapter 4.

5.1 Dynamic CVaR Risk measures on RL application

In this section, we present three distinct experiments that apply dynamic CVaR risk measures within reinforcement learning frameworks. The method in [12] serves as our baseline for comparison throughout all three experiments. We start with the Statistical Arbitrage example, replicating one of the experiments conducted in [12]. Then, we will extend it to a new experiment where we consider the jump cases with different jump frequencies. In addition, we apply our method to a typical stock trading environment which includes an initial stock price and trades constrained by the amount of money rather than the inventory size. The first two experiments are trained and tested on a GPU (NVIDIA GeForce RTX™ 4080 Laptop GPU), while the final experiment is conducted on a CPU (13th Gen Intel® Core™ i9-13900HX × 32).

5.1.1 Statistical Arbitrage

In a RL model, the agent starts each episode with zero inventory. At each timestep $t \in [0, T - 1]$, the agent determines a quantity $|a_t|$ shares of an asset to trade based on

the current price X_t and the current inventory holding q_t . A selling action is defined as $a_t < 0$, while a buying action is defined as $a_t > 0$. If the agent chooses to do nothing, then $a_t = 0$. Similarly, a positive q_t represents the number of shares held by the agent, while a negative q_t indicates the agent owes $|q_t|$ shares to the market. When $q_t = 0$, it indicates that no shares are currently held. We set a fixed number a_{max} to limit the amount the agent can buy or sell in a single timestep. Specifically, the agent can buy up to a_{max} shares or sell up to a_{max} shares. Therefore, the action a_t lies within the interval $(-a_{max}, a_{max})$. Similarly, we impose a constraint on the inventory holding size, denoted by q_{max} . Thus, the inventory holding q_t is controlled within the interval $(-q_{max}, q_{max})$.

The price of the asset follows an Ornstein-Uhlenbeck process[29], which can be expressed as:

$$dX_t = \kappa(\mu - X_t)dt + \sigma dW_t, \quad (5.1.1.1)$$

where μ denotes the mean, κ is the rate of reversion that governs how quickly the asset price reverts to μ , W_t is a Wiener process[43], and σ represents the volatility of the asset.

As introduced in Chapter 3, the objective function in this RL model is defined by equation (3.2.1.4). The goal of the agent is to learn an action selection strategy that minimizes the dynamic CVaR risk measure value of the total costs, denoted as $\tilde{\rho} \left(\sum_{t=0}^{T-1} c_t \right)$. The expression for $\tilde{\rho}$ is presented in equation (3.1.3.10). We adapt the same formula for the cost variable c_t from [12]. The mathematical expression for c_t at each timestep t is given as follows:

$$c_t = - \left(-a_t X_t - \phi(a_t)^2 \right), \quad \text{for } t \in [0, T - 1], \quad (5.1.1.2)$$

where ϕ denotes the coefficient associated with transaction costs.

At $t = T - 1$, trading is halted at the next timestep T , and the terminal cost c_{T-1} is calculated as[12]:

$$c_{T-1} = - \left(-a_{T-1} X_{T-1} - \phi(a_{T-1})^2 \right) - (q_T X_T - \psi q_T^2), \quad (5.1.1.3)$$

where ψ represents the terminal penalty coefficient.

Choice of parameters

Throughout the examples, we consider a single stock with a volatility $\sigma = 0.2$ and a mean $\mu = 1$. The rate of mean reversion is set to $\kappa = 2$. The transaction cost coefficient is fixed at $\phi = 0.005$.

Parameter	Value
Volatility	$\sigma = 0.2$
Mean	$\mu = 1$
Rate of mean reversion	$\kappa = 2$
Trading horizon	$T = 1$
Transaction cost	$\phi = 0.005$
VaR confidence	$p = 80\%$
CVaR confidence	$p = 80\%$
Number of trajectories for V network	$N_V = 500$
Number of trajectories for policy network	$N_{\text{policy}} = 500$
Number of transitions for policy network	$M_{\text{policy}} = 500$
Learning rate of V	$\alpha = 1 \times 10^{-3}$
Learning rate of policy	$\beta = 1 \times 10^{-3}$

Table 5.1: Parameter values used for example 5.1.1.

The V network is initialized with a learning rate of $\alpha = 1 \times 10^{-3}$, and the policy network is initialized with a learning rate of $\beta = 1 \times 10^{-3}$. The trading horizon is set to $T = 1$. The selected value for the total number of training epochs l for the entire algorithm depends on the number of time periods (N). As N increases, the length of the generated trajectories also increases, which not only leads to exponentially higher computational costs for generating states but also requiring more training loops to converge. Table 5.1 summarizes this information, and the parameter settings are consistent with those in [12].

Result

We use the model from [12] as the baseline for comparison with our model. All models in the following test cases are trained using a CVaR risk measure with a confidence level of 0.2. For a given price path, the portfolio value at $t = T$ is calculated as terminal wealth. However, evaluating a model’s performance based on the terminal wealth outcome from one single price path can be misleading. For example, a strategy might perform well in one favorable scenario but could lead to significant losses in other less favorable cases, which would not be captured by just one outcome. Thus, to have a more robust performance analysis, we simulate 30,000 price paths and generate the terminal wealth distributions for both the baseline and our models based on the 30,000 outcomes. The analysis of these distributions, including the mean, standard deviation (Std), VaR, and CVaR, as well as the total training time required for each model, will be presented. Initially, we test under the

same conditions in [12] and then extend to two separate cases: with the terminal penalty ($\psi > 0$) and without the terminal penalty ($\psi = 0$). For each case, we present the results for increasing number of decision timesteps (N) and inventory size (Q).

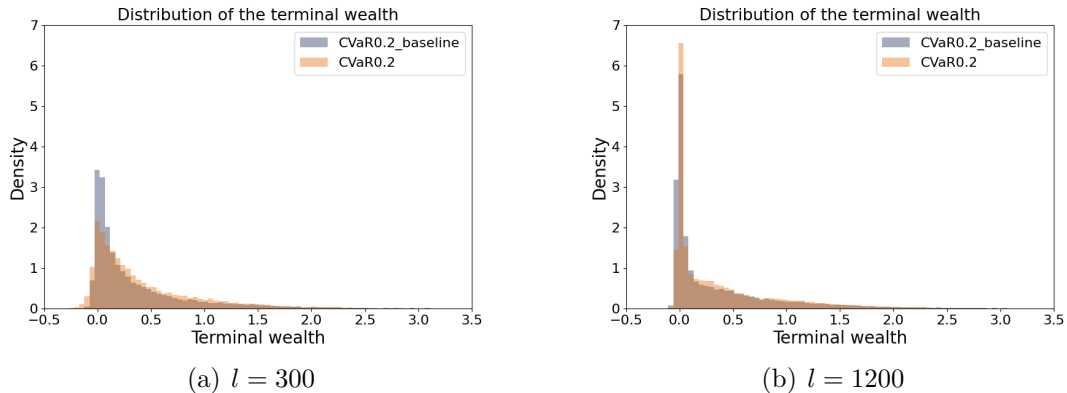


Figure 5.1: Terminal wealth distributions for training loops $l = 300, 1200$ with $\psi = 0.05$

Curve	Mean	Std	VAR (80%)	CVAR (80%)	Time
5.5.2_baseline_300loop	0.3547	0.5005	0.0213	-0.0082	109.7079s
5.5.2_300loop	0.4205	0.5160	0.0330	-0.0230	71.6285s
5.5.2_baseline_1200loop	0.3482	0.5469	-0.0038	-0.0213	435.8855s
5.5.2_1200loop	0.3912	0.5542	0.0062	-0.0086	276.2237s

Table 5.2: Comparison of Statistics for training loops $l = 300, 1200$ with $\psi = 0.05$

In the first experiment, we use the parameters as in [12], where $N = 6$, $q_{max} = 5$, and $a_{max} = 2$. There is a terminal penalty, and we set $\psi = 0.05$. We will compare our model with theirs under the same conditions.

We consider two different values for the number of training loops: $l = 300$ and $l = 1200$. Figure 5.1(a) presents the distributions of the test results for the baseline and our model with $l = 300$, while Figure 5.1(b) shows the distributions of the test results for $l = 1200$. The baseline distribution is shown in blue, and ours is shown in orange. The x-axis is the terminal wealth value. The y-axis represents the probability density value of this distribution. This means that the area under the curve for each distribution sums to 1, allowing for a normalized comparison between the baseline and our model. As shown in Table 5.2, for the case of 300 loops, our result has a better mean and worse CVaR than

the baseline one. For 1200 loops, our model outperforms the baseline in both mean and CVaR. Additionally, no matter in which test case, our model completes training in nearly less than half the time compared to the baseline models. We observe that for the baseline method, the larger training loop results in a worse outcome in both the mean and CVaR. However, for our method, the larger training loop results in a better CVaR and a slightly worse mean. Overall, the baseline model trained with $l = 300$ achieves the best CVaR, while our model trained with $l = 300$ obtains the best mean.

In real-world market trading scenarios, a longer N is typically preferred over a value as short as 6. Additionally, the inventory size is usually larger than 5, and the trading volume per step is often greater than 2. The work in [12] considers only the above test case, which imposes overly restrictive conditions on inventory size, action size, and the time period. To explore more comprehensive scenarios, we increase N from 6 to 11, and then to 21, while also expanding q_{max} from 5 to 100, and a_{max} from 2 to 10. The following six test cases illustrate these expanded scenarios:

1. $N = 6, q_{max} = 100, a_{max} = 10, \psi = 0.05, l = 600,$
2. $N = 11, q_{max} = 100, a_{max} = 10, \psi = 0.05, l = 900$
3. $N = 21, q_{max} = 100, a_{max} = 10, \psi = 0.05, l = 2400,$
4. $N = 6, q_{max} = 100, a_{max} = 10, l = 600,$
5. $N = 11, q_{max} = 100, a_{max} = 10, l = 900,$
6. $N = 21, q_{max} = 100, a_{max} = 10, l = 2400.$

Curve	Mean	Std	VAR (80%)	CVAR (80%)	Time
CVaR0.2_baseline_N6	0.4844	1.9467	-0.9196	-1.5839	213.5062s
CVaR0.2_N6	1.2853	1.7745	0.0156	-0.2318	135.8017s
CVaR0.2_baseline_N11	-1.9591	2.1606	-3.4742	-4.0714	611.0105s
CVaR0.2_N11	0.4986	3.5635	-2.0448	-2.7652	341.4316s
CVaR0.2_baseline_N21	-505.3289	23.2900	-525.1646	-537.9521	3143.9254s
CVaR0.2_N21	-504.6711	23.2900	-524.4551	-537.3021	1670.3865s

Table 5.3: Comparison of Statistics for $N = 6, 11, 21$ with $\psi = 0.05$

Figure 5.2 shows a comparison of the terminal wealth distributions between the baseline model and our model with a terminal penalty of $\psi = 0.05$. The statistics for each

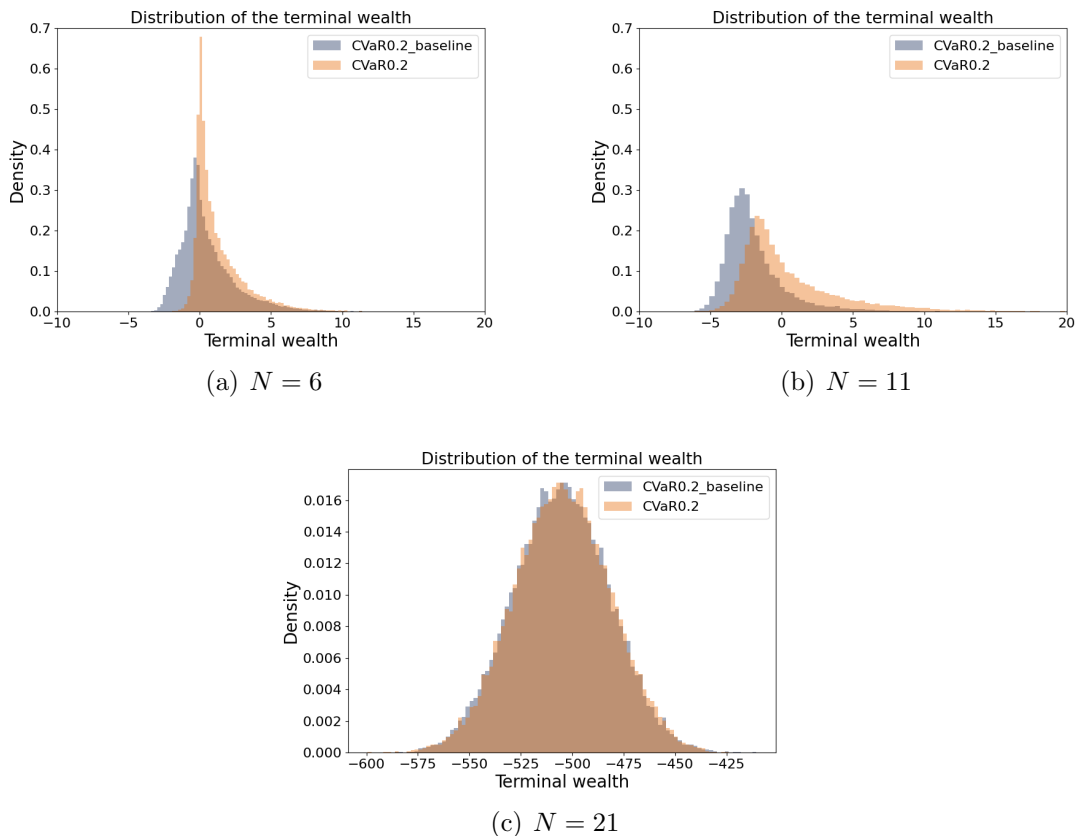


Figure 5.2: Terminal wealth distributions for $N = 6, 11, 21$ with $\psi = 0.05$

distribution are provided in Table 5.3. Figure 5.2(a) represents the case with $N = 6$, and Figure 5.2(b) shows the case with $N = 11$. In both cases, our method consistently produces a distribution with a better mean and fewer losses. It is also worth noting that the training time for our method is still approximately half that of the baseline. Figure 5.2(c) illustrates the $N = 21$ case, where both the baseline and our model perform poorly, exhibiting high negative means and standard deviations. Despite increasing the training loops to 2400, the results remain non-convergent. This suggests a potential limitation in the baseline method when dealing with larger N , inventory size, and action size, along with a terminal penalty. Since our method is based on the baseline method, it shares the same issue.

We then test the models under the condition where $\psi = 0$; i.e. there is no terminal

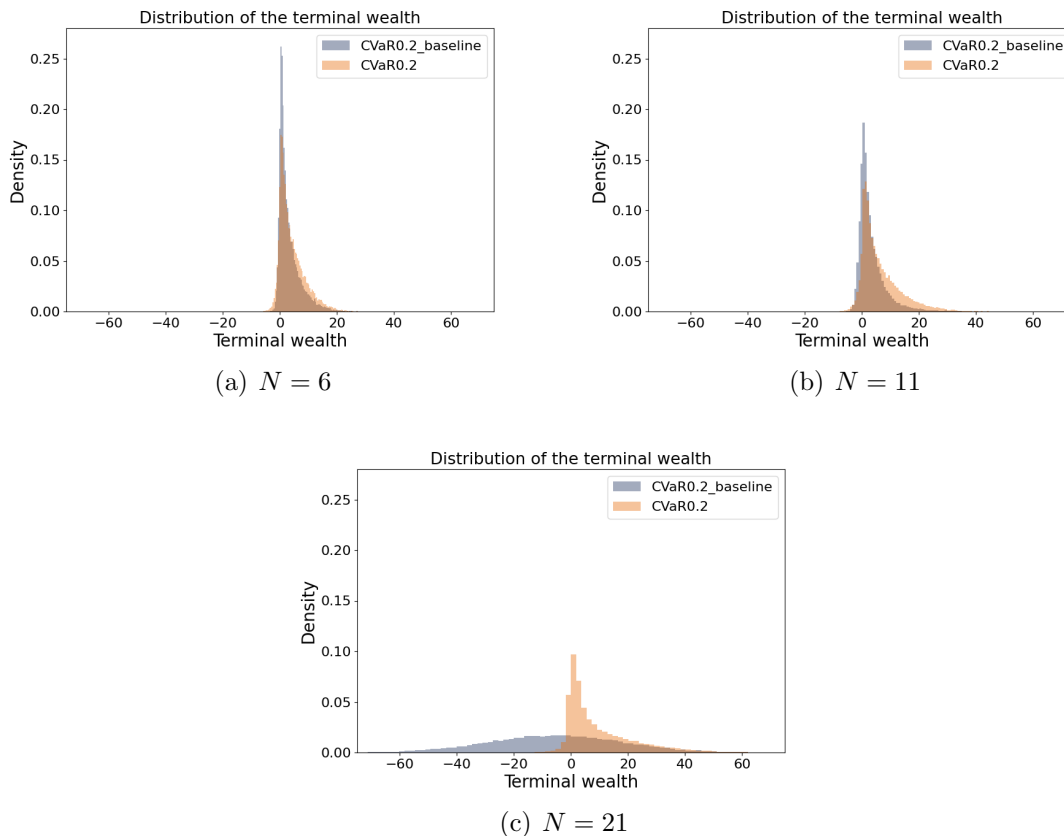


Figure 5.3: Terminal wealth distributions for $N = 6, 11, 21$ with $\psi = 0$

Curve	Mean	Std	VAR (80%)	CVAR (80%)	Time
CVaR0.2_baseline_N6	3.1533	3.8557	0.3657	-0.1980	215.8686s
CVaR0.2_N6	4.1485	4.5667	0.5391	-0.3797	137.8757s
CVaR0.2_baseline_N11	3.3624	5.0483	0.0649	-0.9294	617.2624s
CVaR0.2_N11	6.4780	7.1249	1.0717	-0.2419	314.0352s
CVaR0.2_baseline_N21	-5.3289	23.2890	-25.1646	-37.9521	3187.4091s
CVaR0.2_N21	9.6354	11.9256	0.7259	-0.8641	1594.2923s

Table 5.4: Comparison of Statistics for $N = 6, 11, 21$ with $\psi = 0$

penalty. The statistical data are provided in Table 5.4. In the case when $N = 6$, the baseline model exhibits a slightly higher CVaR, while our model shows better performance in

terms of the mean. As the trajectory length increases to 11 and then to 21, our model significantly outperforms the baseline in both mean and CVaR. The entire distribution of our results shifts to the right compared to the baseline, with the shift becoming more prominent as N increases. This indicates a substantial improvement in both performance and risk management. Notably, the time required for our method remains approximately half of the baseline. Another observation is that as the length of the trajectories increases, the standard deviation of the distributions becomes larger for both models. This is expected, as longer trajectories typically involve more variability due to the increased influence of random market fluctuations.

5.1.2 Jump Cases

In financial modeling, a *jump* refers to a sudden and significant change in the value of an asset that cannot be captured by standard continuous price evolution models, such as the Ornstein-Uhlenbeck process used in equation (5.1.1.1). These jumps can either increase or decrease the asset's price. Incorporating jumps into a model allows for a more accurate representation of real-world market dynamics, where such abrupt changes may occur due to events such as economic announcements[10] or geopolitical developments[37]. To further demonstrate the robustness of our approach, we extend our analysis to include scenarios characterized by market jumps, focusing specifically on upward jumps.

Let J denote the jump component in our model, with the occurrence of jumps defined as dJ . The process dJ is modeled using a Poisson process. Specifically, dJ over a small time increment τ is determined by:

$$dJ \sim \text{Poisson}(\Lambda \cdot \tau),$$

where Λ is the rate parameter of the Poisson process, representing the expected number of jumps per unit time.

We use a logistic function to ensure that the jump magnitudes are smoothly distributed within a specified range $[J_{min}, J_{max}]$. The jump magnitude is calculated as:

$$J = \left(J_{min} + (J_{max} - J_{min}) \cdot \frac{1}{1 + \exp(-x)} \right) \cdot dJ, \quad x \sim \mathcal{N}(0, 1),$$

where x is drawn from a normal distribution with a mean of 0 and a standard deviation of 1. Here, J_{min} represents the minimum magnitude of the jump and J_{max} represents the maximum magnitude of the jump. The price evolution then becomes:

$$dX_t = \kappa(\mu - X_t)dt + \sigma dW_t + J. \tag{5.1.2.1}$$

By changing the values of $[J_{min}, J_{max}]$ and λ , we can control the frequency and magnitude of the simulated jumps.

Result

In this experiment, we fix $N = 10+1$ and do not consider the terminal penalty. The training loop is set to $l = 900$. We train our model and the baseline model under two different values of λ with two different ranges of $[J_{min}, J_{max}]$. Given that the mean asset price is defined as 1, the pairs (J_{min}, J_{max}) are set to $(0.03, 0.06)$ and $(0.05, 0.10)$, representing price fluctuations of 3% to 6% and 5% to 10%,

We train the models under various jump conditions. The details of all four jump cases are described below:

1. $J_{min} = 0.05, J_{max} = 0.1, \lambda = 0.005$,
2. $J_{min} = 0.05, J_{max} = 0.1, \lambda = 0.05$.
3. $J_{min} = 0.03, J_{max} = 0.06, \lambda = 0.005$,
4. $J_{min} = 0.03, J_{max} = 0.06, \lambda = 0.05$.

Curve	0 jump case	2 jumps case	4 jumps case
CVaR0.2_baseline_ $\lambda = 0$ Time: 617.2624s	mean: 3.3624 cvar: -0.9294	mean: 3.7698 cvar: -0.5564	mean: 5.3770 cvar: 0.8296
CVaR0.2_ $\lambda = 0$ Time: 314.0352s	mean: 6.4780 cvar: -0.2419	mean: 7.0894 cvar: 0.0927	mean: 7.0934 cvar: -1.3238
CVaR0.2_baseline_ $\lambda = 0.005$ Time: 656.2229s	mean: 5.8226 cvar: -0.1317	mean: 6.3974 cvar: 0.2310	mean: 6.5966 cvar: -0.6904
CVaR0.2_ $\lambda = 0.005$ Time: 374.5136s	mean: 6.2207 cvar: -0.3212	mean: 6.6267 cvar: 0.1927	mean: 7.0612 cvar: -0.1023
CVaR0.2_baseline_ $\lambda = 0.05$ Time: 638.2356s	mean: 5.3269 cvar: -0.4222	mean: 6.1933 cvar: 0.1021	mean: 6.4668 cvar: -0.7894
CVaR0.2_ $\lambda = 0.05$ Time: 363.4547s	mean: 6.5223 cvar: -0.3754	mean: 6.9385 cvar: 0.0914	mean: 7.1611 cvar: -0.5671

Table 5.5: Comparison of Statistics for $\lambda = 0, 0.005, 0.05$ with jumps in $[0.05, 0.1]$

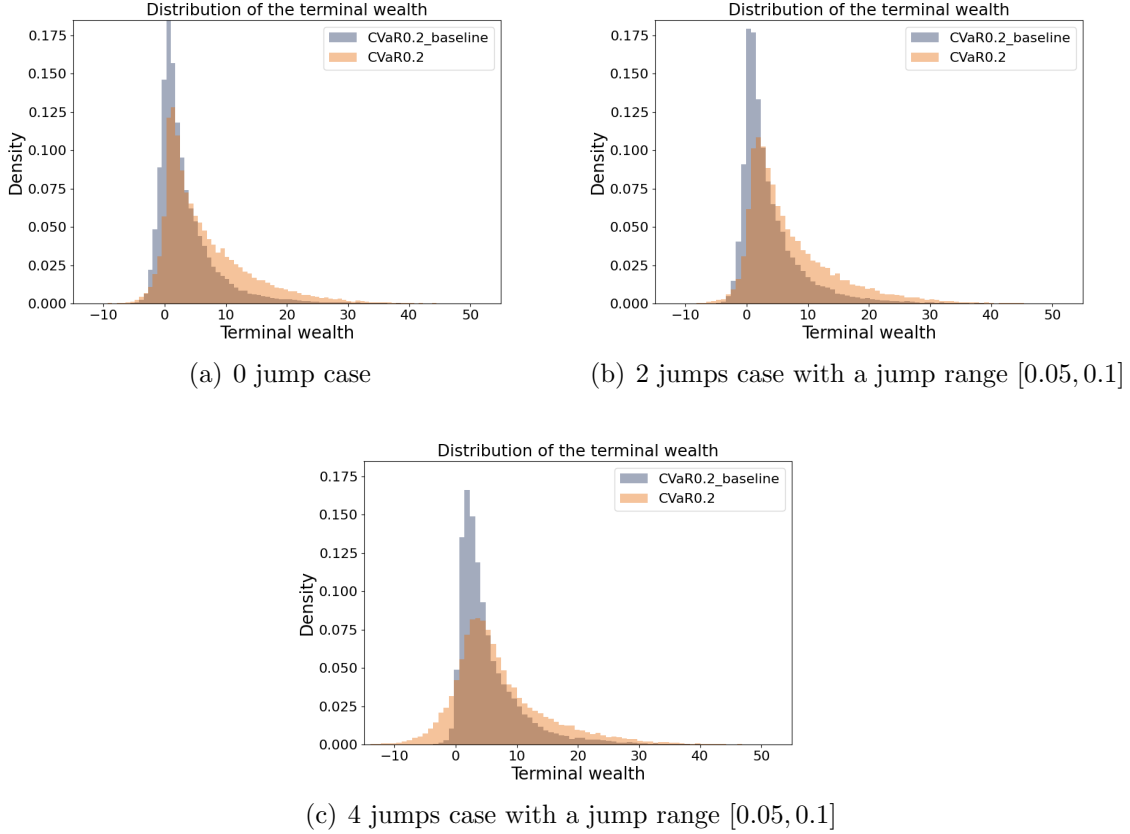


Figure 5.4: Performance of the model trained with $\lambda = 0$

We then compare the performance of the models trained with and without jumps across three different test sets: price paths without any jumps, price paths with two jumps, and price paths with four jumps.

We first fix the jump size with $J_{min} = 0.05$, and $J_{max} = 0.1$, and train the model under three conditions: no jumps, $\lambda = 0.005$, and $\lambda = 0.05$. Table 5.5 compares the mean and CVaR (80%) values across different scenarios, with the model trained under varying risk levels ($\lambda = 0, 0.005, 0.05$) and tested under different jump occurrences (0, 2, and 4 jumps). Focusing on the rows of Table 5.5, we begin by comparing the trained models across different test cases. For $\lambda = 0$, our model achieves better mean and CVaR values in both the 0 jump and 2 jumps cases compared to the baseline. In the 4 jumps case, our model obtains a better mean, but the CVaR result is worse.

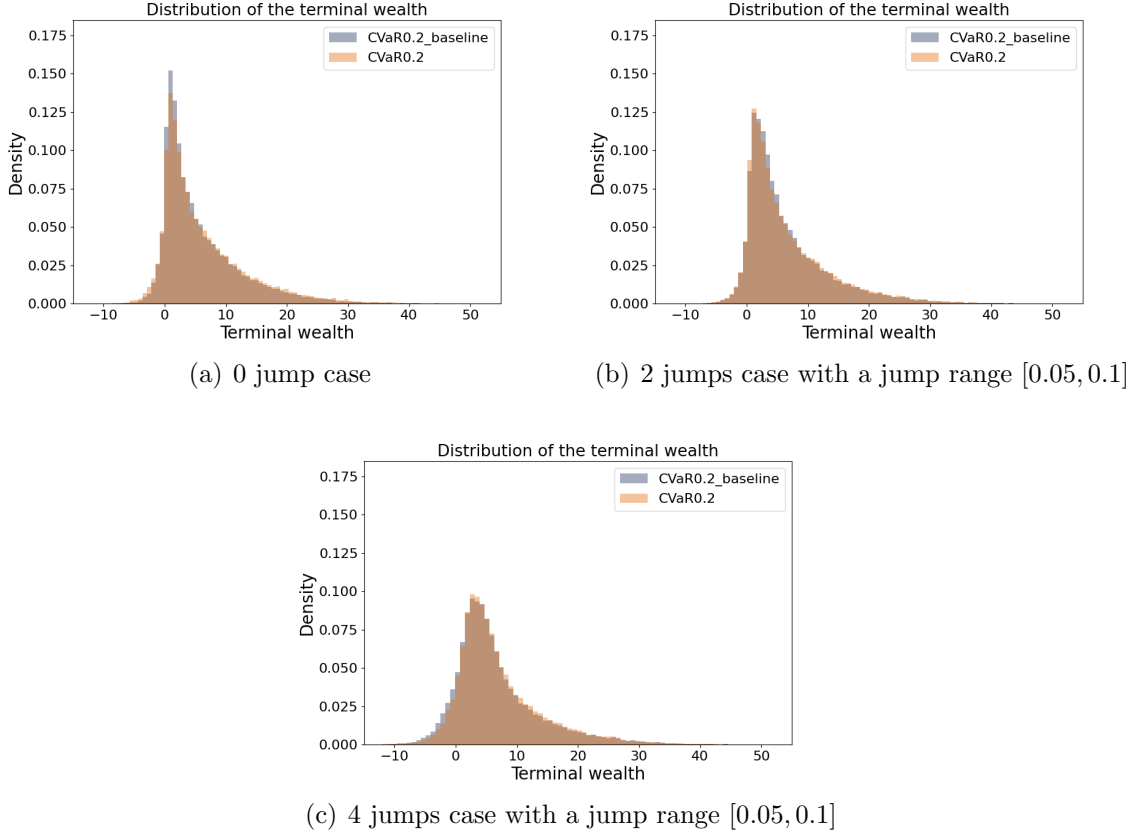


Figure 5.5: Performance of the model trained with $\lambda = 0.005$

Figure 5.4 illustrates corresponding the distribution plots for models trained with no jump: Subplot (a) shows the distributions tested with no jumps, Subplot (b) shows the distributions tested with 2 jumps, and Subplot (c) shows the distributions tested with 4 jumps. In all three cases, the distribution of our model consistently shifts to the left compared to the baseline, resulting in a better mean outcome. However, as seen in Subplot (c), our model exhibits poor tail performance, as shown by the worse CVaR in this scenario.

We also evaluate scenarios where a jump occurs during the training process, testing both a low probability case with $\lambda = 0.005$ and a higher probability case with $\lambda = 0.05$. Figures 5.5 and 5.6 illustrate the performance of models trained under these respective settings. In both figures, the distributions of our model and the baseline nearly overlap, making it difficult to distinguish between them visually. However, examining the data in

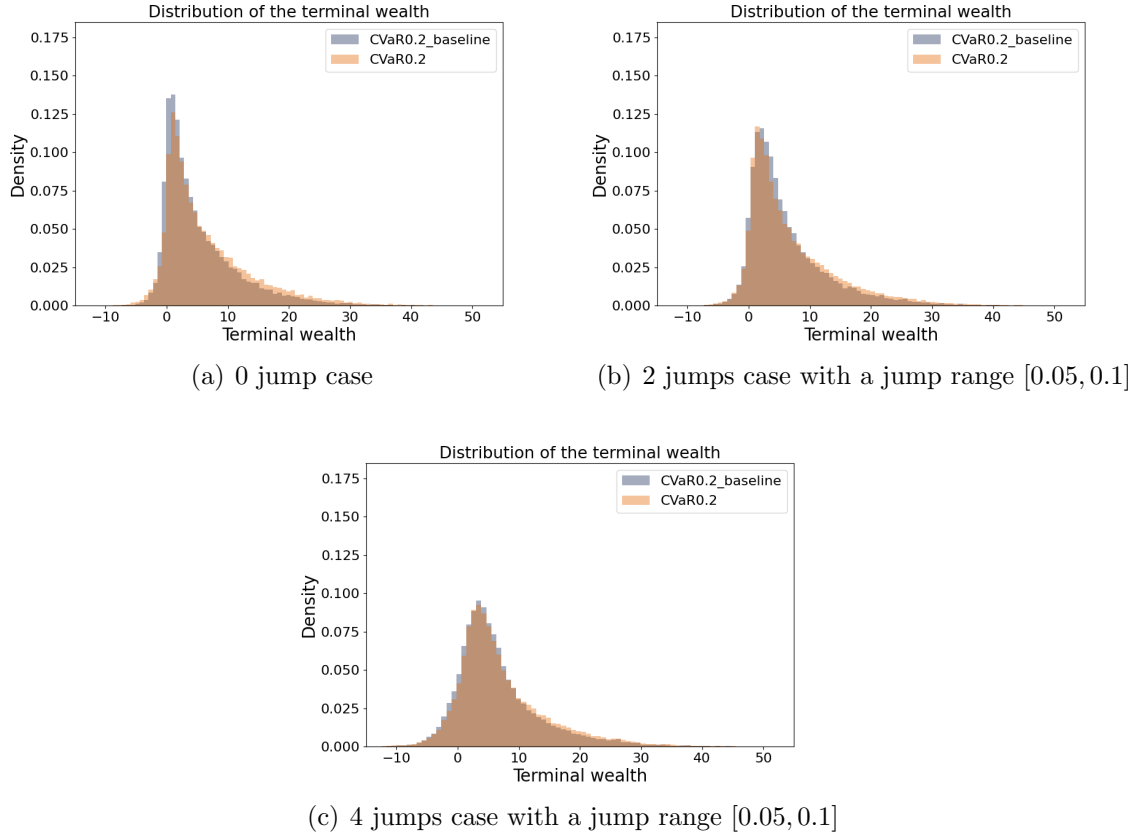


Figure 5.6: Performance of the model trained with $\lambda = 0.05$

Table 5.5, we observe that for both $\lambda = 0.005$ and $\lambda = 0.05$, our model exhibits worse CVaR performance in the test case without jumps, but outperforms the baseline in CVaR when jumps occur. This contrasts with the $\lambda = 0$ case. Next, we compare the performance of different models under the same test cases by examining the columns in Table 5.5. For the 0 jump test case, our model trained without jumps achieves the best CVaR, while our model trained with $\lambda = 0.05$ achieves the best mean. In the 2 jumps test case, our model trained without jumps achieves the best mean, while the baseline model trained with $\lambda = 0.005$ achieves the best CVaR. In the 4 jumps test case, our model trained with $\lambda = 0.05$ achieves the best mean, while the baseline model trained with $\lambda = 0$ achieves the best CVaR.

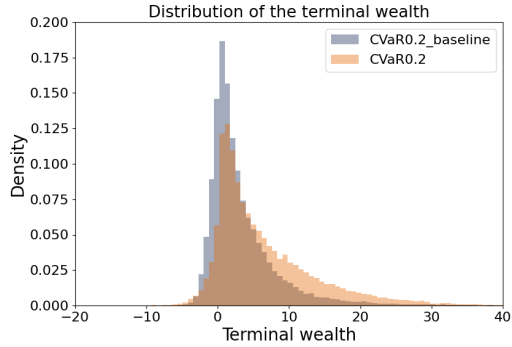
Finally, by examining the diagonal entries in Table 5.5, we observe that in the case

where models are trained and tested without any jumps, as well as in the case where models are trained and tested with the highest λ value ($\lambda = 0.05$), our model outperforms the baseline in both mean returns and CVaR. Additionally, in the case where models are trained and tested with a moderate likelihood of jumps ($\lambda = 0.005$), our model achieves better mean returns and only slightly worse CVaR. In conclusion, our models consistently outperform the baseline models in terms of mean returns across all test sets and achieve better CVaR in most test cases across different λ settings. Moreover, the time required to train our models is approximately half that of the baseline models.

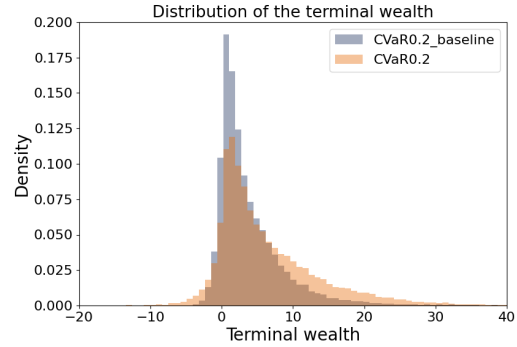
Curve	0 jump case	2 jumps case	4 jumps case
CVaR0.2_baseline_ $\lambda = 0$ Time: 617.2624s	mean: 3.3624 cvar: -0.9294	mean: 3.9202 cvar: -0.2560	mean: 4.6437 cvar: 0.5457
CVaR0.2_ $\lambda = 0$ Time: 314.0352s	mean: 6.4780 cvar: -0.2419	mean: 6.2933 cvar: -0.6455	mean: 4.7588 cvar: -4.7754
CVaR0.2_baseline_ $\lambda = 0.005$ Time: 656.2229s	mean: 5.7675 cvar: -0.0945	mean: 5.7096 cvar: -0.3477	mean: 4.3601 cvar: -3.5428
CVaR0.2_ $\lambda = 0.005$ Time: 374.5136s	mean: 6.2207 cvar: -0.3194	mean: 6.5255 cvar: 0.0057	mean: 6.1067 cvar: -2.1672
CVaR0.2_baseline_ $\lambda = 0.05$ Time: 638.2356s	mean: 5.7347 cvar: -0.2118	mean: 5.4408 cvar: -0.7888	mean: 3.8188 cvar: -4.3626
CVaR0.2_ $\lambda = 0.05$ Time: 363.4547s	mean: 6.2723 cvar: -0.2558	mean: 6.5028 cvar: -0.0440	mean: 5.9033 cvar: -2.5013

Table 5.6: Comparison of Statistics for $\lambda = 0, 0.005, 0.05$ with jumps in $[0.03, 0.06]$

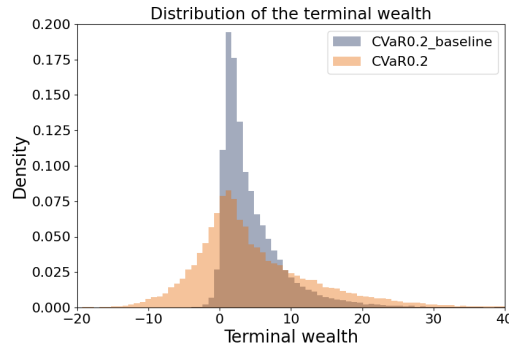
For the test cases with a smaller jump range, $J_{min} = 0.03$ and $J_{max} = 0.06$, the statistical results are presented in Table 5.6, and the corresponding distribution plots are shown in Figures 5.7, 5.8, and 5.9. The test results exhibit a trend similar to those observed with a larger jump range. Specifically, in the diagonal entries of Table 5.6, where each model is tested under its corresponding training conditions, our model achieves better mean returns and CVaR compared to the baseline model. However, there are some differences when compared to the test cases with a larger jump range. For the case where models are trained with $\lambda = 0.05$, our model shows a worse CVaR than the baseline. Nevertheless, when the model is trained with the same λ but with a larger jump size, our model performs better in terms of risk management. Additionally, when models are trained without any jumps, the baseline model demonstrates better CVaR than ours when tested under both the 2 jumps and 4 jumps cases. This discrepancy is not observed when testing with a larger jump range.



(a) 0 jump case



(b) 2 jumps case with a jump range $[0.03, 0.06]$



(c) 4 jumps case with a jump range $[0.03, 0.06]$

Figure 5.7: Performance of the model trained with $\lambda = 0$

Regardless of the jump range, our model consistently demonstrates superior mean returns and, in most instances, better CVaR values compared to the baseline. When the models are trained and tested under the same conditions, our model achieves better or comparable CVaR values relative to the baseline. Furthermore, even when tested under unmatched jump scenarios, our model generally outperforms the baseline in both mean returns and CVaR. Based on these test results, we can conclude that our method is more robust than the baseline.

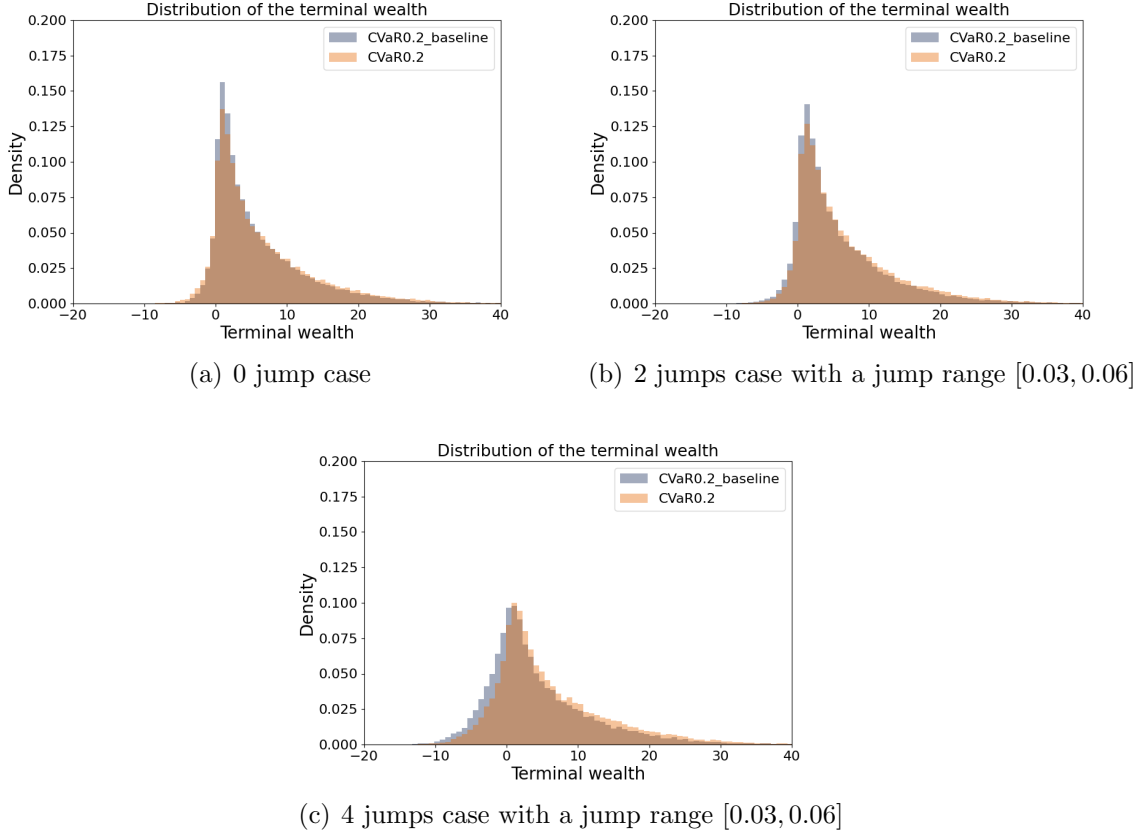


Figure 5.8: Performance of the model trained with $\lambda = 0.005$

5.1.3 Stock Trading

We observe that the experiments in Sections 5.1.1 and 5.1.2 are designed to explore specific trading scenarios where the initial stock price is not fixed at a particular X_0 value, and the trading process is limited by the inventory size q_{max} , rather than the amount of money available. These setups serve as valuable case studies for certain trading environments. In this section, we introduce a more conventional stock trading scenario, featuring a fixed initial price value and an initial amount of money. This allows us to explore the trading dynamics in a context more closely aligned with traditional financial markets, which will be further discussed.

In all examples, we consider a single stock with an initial market price X_0 . The number of shares held at timestep t is denoted by q_t , and the amount of money on hand at timestep

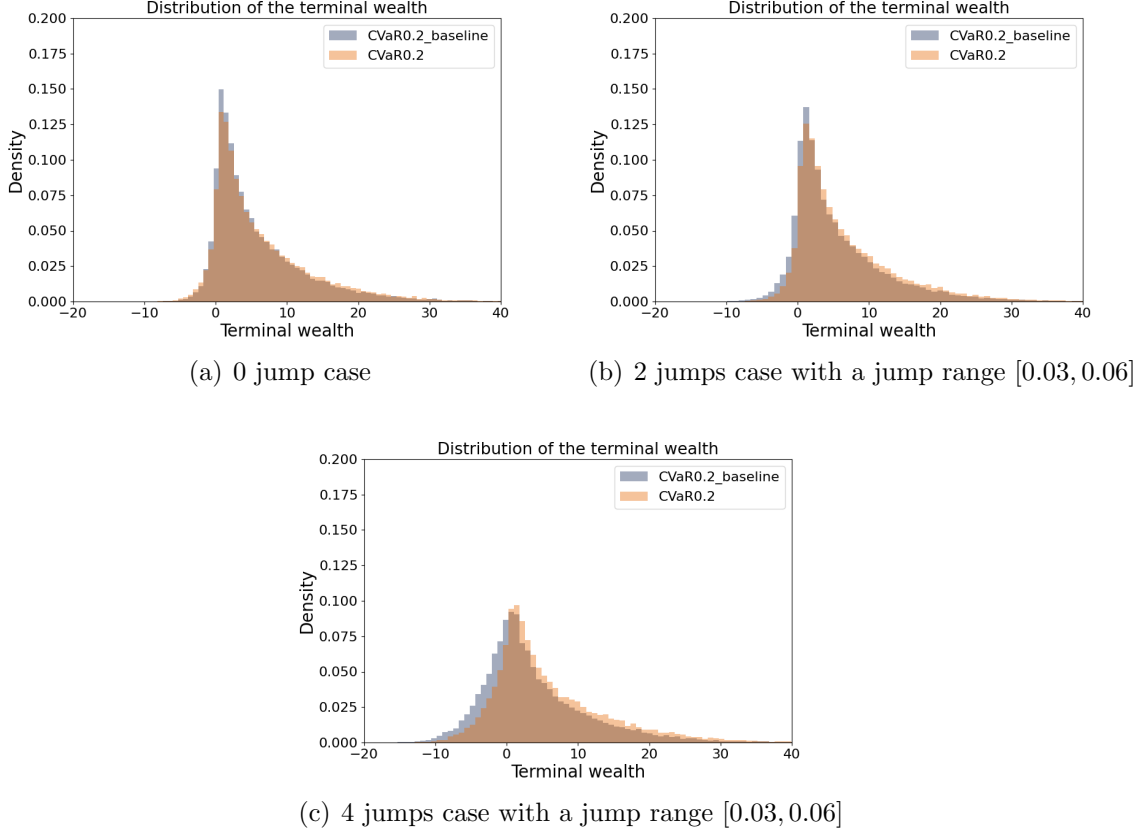


Figure 5.9: Performance of the model trained with $\lambda = 0.05$

t is denoted by m_t . The agent starts with q_0 shares and m_0 amount of money, resulting in an initial portfolio value of $X_0q_0 + m_0$. At each timestep $t \in T$, the agent executes a trade with a quantity $a_t \in (-q_t, a_{\text{available}})$, where $a_{\text{available}}$ represents the maximum number of shares that can be bought given the current price X_t and money holding m_t . Trades are selected by the RL agent, which is based on the state s_t composed by (X_t, q_t, m_t, t) . It is important that both m_t and q_t must be non-negative, meaning that short selling and borrowing money are not permitted in this case.

The price of the asset continues to follow the Ornstein-Uhlenbeck process described in section 5.1.1. The cost c_t is defined as the negative value of the profit obtained in each state, such that minimizing the cumulative cost essentially maximizes the total profit, leading to better terminal wealth. For $t = 1, \dots, T - 1$, the cost c_t is determined by the

change in portfolio value between time t and the next timestep $t + 1$. The portfolio value at any timestep t is given by $y_t = q_t X_t + m_t$. The asset price at the next timestep, X_{t+1} , is obtained from equation (5.1.1.1). The inventory and money holdings at the next timestep are updated as follows:

$$\begin{aligned} q_{t+1} &= q_t + a_t, \\ m_{t+1} &= m_t - (a_t X_t + \phi(a_t)^2), \end{aligned}$$

where ϕ is the transaction cost coefficient. Thus, the cost c_t for $t \neq T$ is expressed as:

$$c_t = y_t - y_{t+1}. \tag{5.1.3.1}$$

Choice of parameters

We adopt the parameter settings from Table 5.1 for this experiment. The initial market price of the stock is $X_0 = 1$, and the agent begins with 100 shares and \$100 in cash, resulting in an initial portfolio value of \$200. The algorithm is trained over $l = 600$ epochs, with $M_{\text{policy}} = 200$ transitions used for the policy update. The number of decision timesteps is fixed at $N = 11$ across all test cases. These additional parameters are provided in Table 5.7 below.

Parameter	Value
Initial stock price	$X_0 = 1$
Initial share holdings	$q_0 = 100$
Initial money holdings	$m_0 = 100$
Number of decision timesteps	$N = 11$
Whole algorithm training loops	$l = 600$
Number of transitions for policy network	$M_{\text{policy}} = 200$

Table 5.7: Parameter values used for example 5.1.3.

Result

Both our model and the baseline model are trained under conditions where no jumps occur during the training process. We test the models in four different scenarios: no jump, up jump, down jump, and mixed jump. Specifically, we test the up jump case with a single

upward jump, the down jump case with three down jumps, and the mixed jump case where jumps can be either up or down, with two jumps triggered. The test results are shown in Table 5.8. Our model not only achieves better mean and CVaR compared to the baseline in the no jump case but also outperforms the baseline in the down jump and mixed jump cases. In the up jump case, our model produces a better mean, though it performs slightly worse in CVaR. However, it is noteworthy that our model achieves a slightly better VaR in this case.

Subplot (a) in Figure 5.10 represents the no jump case, Subplot (b) the up jump case, Subplot (c) the down jump case, and Subplot (d) the mixed jump case. The x-axis and y-axis definitions remain the same as in Section 5.1.1. Subplots (a), (c), and (d) demonstrate our model’s superior performance, with distributions shifting to the right and improved tail behavior. Subplot (b) shows a slightly worse tail performance for our model, this aligns with the test data. Overall, our model outperforms the baseline in mean and CVaR in most cases, demonstrating greater robustness against market jumps.

Curve	Test case	Mean	Std	VAR (80%)	CVAR (80%)
CVaR0.2_baseline	No Jump	-4.9804	5.8773	-9.8888	-13.2305
CVaR0.2	No Jump	-0.6171	8.6529	-7.9375	-12.7138
CVaR0.2_baseline	Up Jump	-1.2655	5.8754	-6.1993	-9.4914
CVaR0.2	Up Jump	1.0688	8.6983	-6.1940	-11.0717
CVaR0.2_baseline	Down Jump	-14.7755	5.9230	-19.7781	-23.0689
CVaR0.2	Down Jump	-5.8793	8.6986	-13.1341	-18.0224
CVaR0.2_baseline	Mixed Jump	-5.2841	5.9028	-10.2551	-13.5420
CVaR0.2	Mixed Jump	-0.2120	8.6899	-7.4761	-12.3387

Table 5.8: Comparison of Statistics for Various Scenarios

Combining the observed experimental results from Sections 5.1.1, 5.1.2, and 5.1.3, we can conclude that in most cases, our model achieves a better mean and CVaR compared to the baseline. Recall that our value functions are computed based on equations (3.2.3.6) and (3.2.3.7), where greedy actions are selected to minimize the expected cost at each state s_t , thereby maximizing total profit and leading to improved mean returns and CVaR performance. However, this method has some drawbacks; for instance, focusing only on the greedy action’s result when evaluating value functions neglects the potential outcomes of other actions, leading to inaccurate state value estimates. As a result, the agent may occasionally select suboptimal actions, increasing the likelihood of getting stuck in local optima. Overall, our model not only outperforms the baseline in terms of mean and CVaR but also significantly reduces training time.

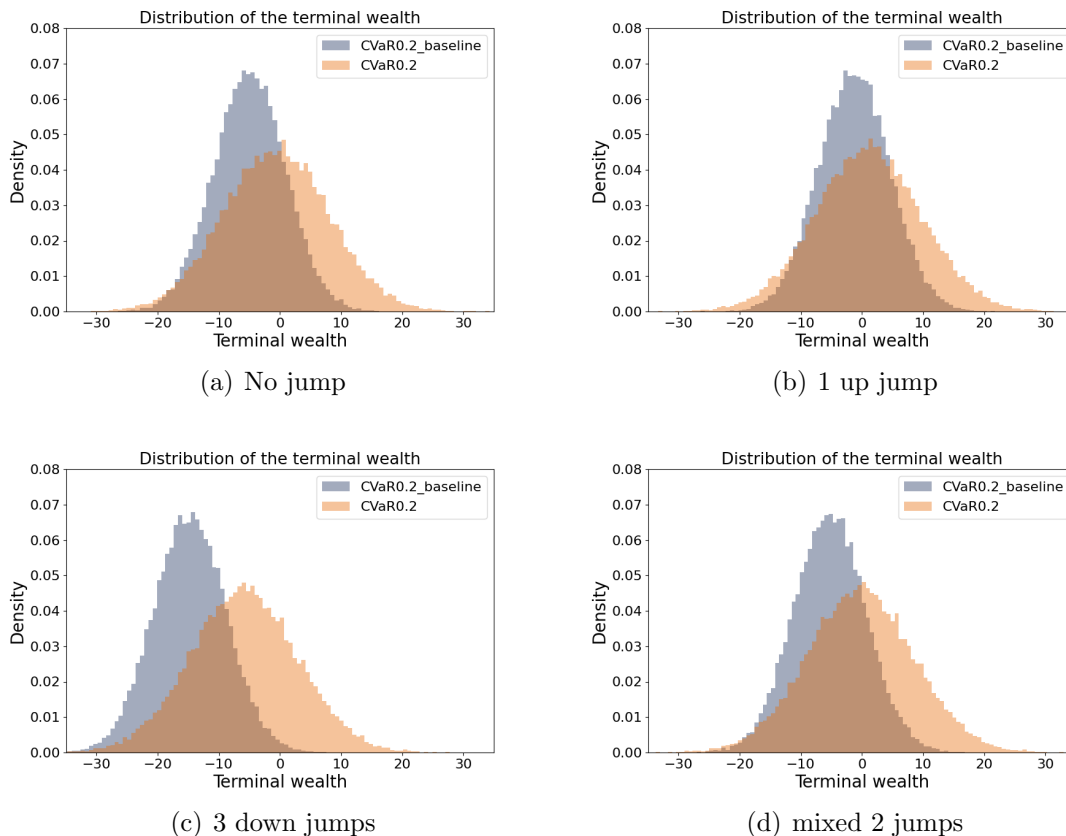


Figure 5.10: Comparison of distribution performance between our model and the baseline across different jump scenarios

5.2 Optimal Trade Execution

In this section, we present two examples related to the optimal trade execution problem. In Section 5.2.1, we analyze the results obtained from comparing our RL model with those from Almgren-Chriss model, which serves as a benchmark for the optimal trade execution problem. In Section 5.2.2, we present the computed results of our RL model incorporating a nonlinear price impact function. Both experiments are conducted on a CPU (13th Gen Intel® Core™ i9-13900HX × 32). Further details of each example are provided below.

5.2.1 Optimal Execution of Portfolio Transactions

We refer to the parameter settings in [3]. We start by holding one million shares of a stock with an initial market price of $X_0 = 50$ \$/share. This stock has an annual volatility of 12%, a bid-ask spread of $\frac{1}{8}$ \$/share, and a median daily trading volume of 5 million shares. With a trading year of 250 days, the daily volatility is calculated as $\frac{0.12}{\sqrt{250}} = 0.0076$. The price evolution is described in equation (4.2.5). The volatility σ used here is computed as the daily volatility multiplied by the initial price, shown as $\sigma = 0.0076 \times 50 = 0.38$. We assume a liquidation time $T = 60$ days. This period is divided into daily trades, so $N = 60$ and we get $\tau = 1$ day. We define γ in the permanent price impact function as 2.5×10^{-7} . For the temporary price impact function, Almgren-Chriss model chooses the fixed transaction cost ϵ as half of the bid-ask spread, which is $\frac{1/8}{2} = 0.0625$. For η , we assume that for each 1% of the daily volume traded, there is a price impact equal to the bid-ask spread. Under this assumption, we have $\eta = \frac{1/8}{0.01 \times 5 \times 10^6} = 2.5 \times 10^{-6}$. For our RL model, the V network is initialized with a learning rate of $\alpha = 1 \times 10^{-3}$, and the policy network is initialized with a learning rate of $\beta = 1 \times 10^{-3}$. All the parameters above are summarized in Table 5.9.

Parameter	Value
Initial stock price	$X_0 = 50$ \$/share
Initial holdings	$Q = 1,000,000$ shares
Annual volatility	12%
Bid-ask spread	$\frac{1}{8}$ \$/share
Daily volatility	$\sigma = 0.38$
Median daily trading volume	5,000,000 shares
Liquidation time	$T = 60$ days
Number of time periods	$N = 60$
Time step	$\tau = 1$ day
Permanent price impact	$\gamma = 2.5 \times 10^{-7}$ \$/share ²
Temporary price impact	$\eta = 2.5 \times 10^{-6}$ (\$/share)/(share/day)
Fixed transaction cost	$\epsilon = 0.0625$ \$/share
Learning rate of V	$\alpha = 1 \times 10^{-3}$
Learning rate of policy	$\beta = 1 \times 10^{-3}$

Table 5.9: Parameter values used in the optimal trade execution example.

In our tests, we consider 13 different values of λ : 1×10^{-6} , 1.5×10^{-6} , 2×10^{-6} , 2.5×10^{-6} , 3×10^{-6} , 3.5×10^{-6} , 4×10^{-6} , 4.5×10^{-6} , 5×10^{-6} , 5.5×10^{-6} , 6×10^{-6} ,

6.5×10^{-6} , and 7×10^{-6} . Each case leads to an optimal trading list $\{u_t\}_{t=0}^{N-1}$ and its corresponding inventory list $\{q_t\}_{t=0}^N$. We perform trade u_t for $t = 0$ to $N - 1$ on 10,000 price trajectories and calculate the value of $E(x) + \lambda Var(x)$ at final time. Note that due to the random process involved in the price evolution, the value of $E(x) + \lambda Var(x)$ may vary even under the same trade list. Sometimes, this value may be larger, while other times it may be smaller, making it challenging to accurately evaluate the performance of the trade list. To mitigate this issue, we test each trading list 100 times over 10,000 price trajectories. This yields a list of 100 values for $E(x) + \lambda Var(x)$, and we use the median value as the final $E(x) + \lambda Var(x)$ for that trade list.

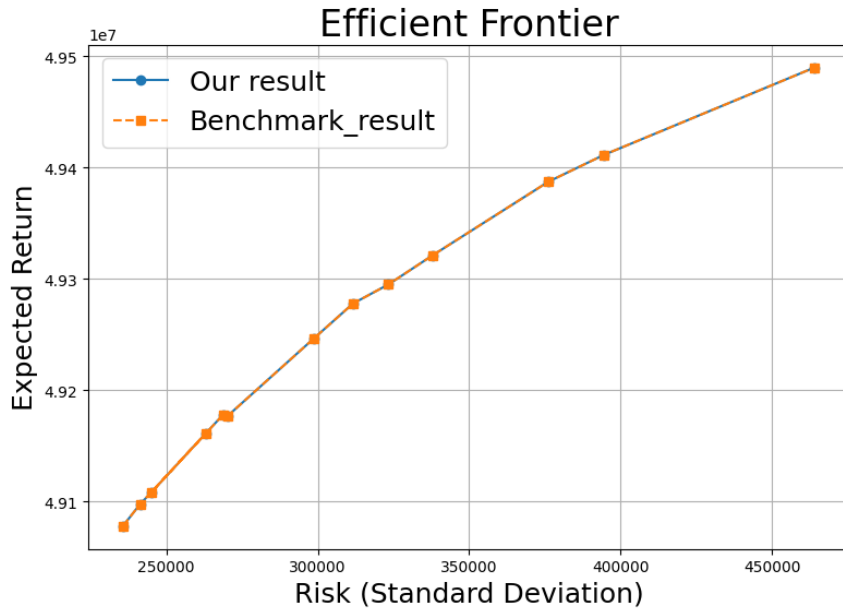


Figure 5.11: Efficient frontier curves for each λ value.

The benchmark value of $E(x) + \lambda Var(x)$ for each λ value is computed using the trading list $\{u_t\}_{t=0}^{N-1}$ obtained from the Almgren-Chriss model (see Chapter 4.3). The trading list $\{u_t\}_{t=0}^{N-1}$ for our test cases is generated through the reinforcement learning model presented in Chapter 4.4.

The optimal trade list that each model finds for each λ produces a corresponding $E(x)$ and $Var(x)$ value. We construct an efficient frontier curve from the list of these $E(x)$ and $Var(x)$ pairs, which has a size of 13. The plot can be found in Figure 5.11. In this plot, the y-axis represents the expected return, which is the initial portfolio value minus the expected shortfall $E(x)$, denoted as $X_0Q - E(x)$. The x-axis represents the risk, shown

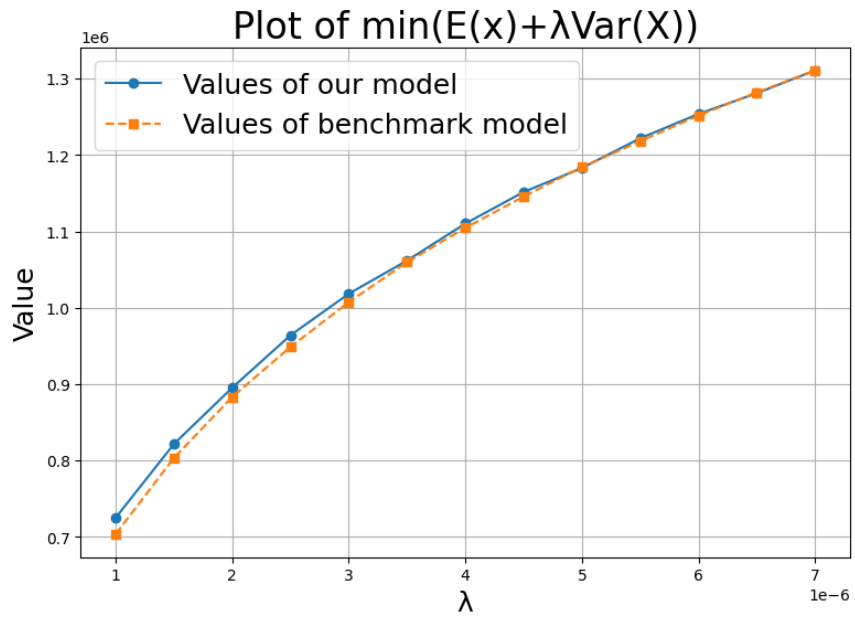


Figure 5.12: $\min(E(x) + \lambda \cdot \text{Var}(x))$ for each test λ case.

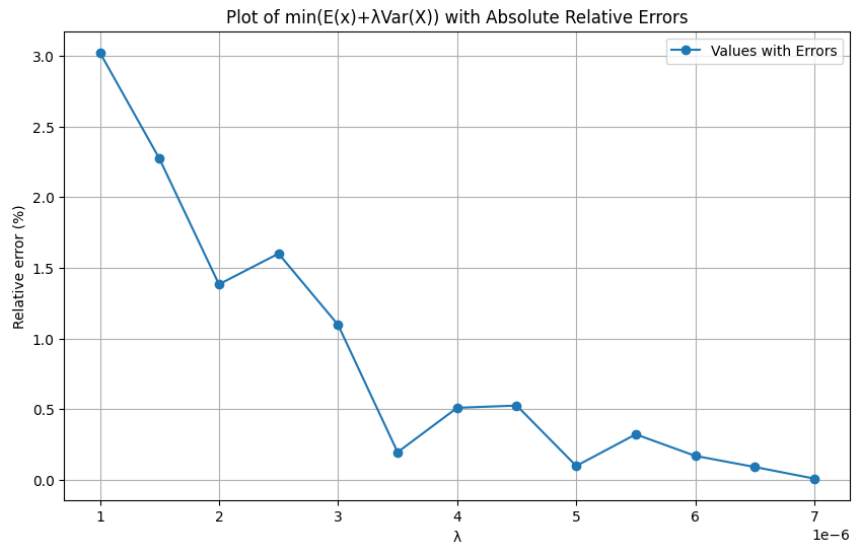


Figure 5.13: Relative error of $E(x) + \lambda \cdot \text{Var}(x)$ for each test λ case.

as the standard deviation, which is the square root of the variance $Var(x)$. The efficient frontier plot illustrates the trade-off between risk and return. Points along the curve represent optimal portfolios that offer the highest possible expected return for a given level of risk. The shape of the curve typically shows that as the expected return increases, so does the associated risk, reflecting the fundamental trade-off in financial optimization. Portfolios lying below the curve are considered suboptimal because they provide lower returns for a given level of risk, while portfolios above the curve are unattainable due to the constraints of the model.

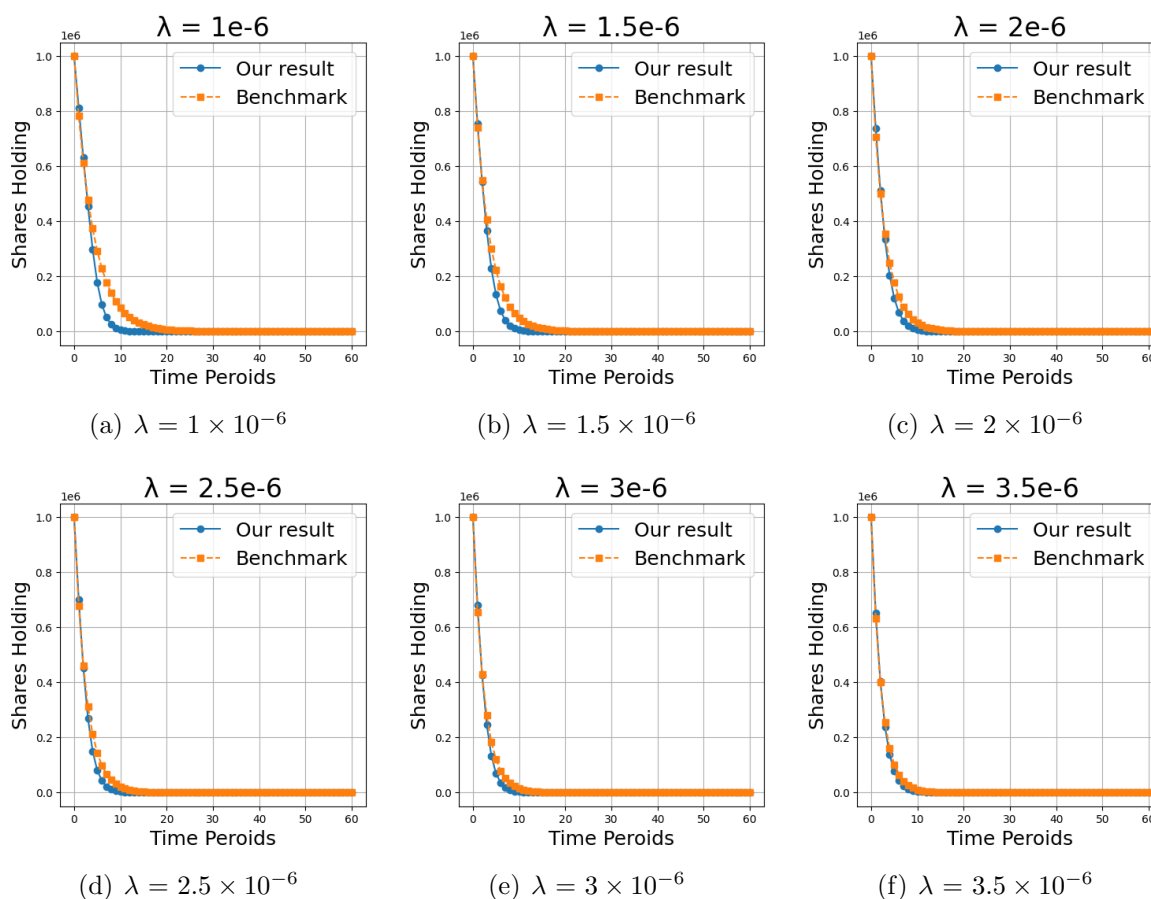


Figure 5.14: Shares Holding vs Time Periods for each test lambda cases.

Our model's curve is in blue while the benchmark one is in orange. However, the values of the expected return and standard deviation of these two models computed for each test case are so close that the two efficient frontier curves overlap, making it difficult to

distinguish between the curves. We consider the plot of the value of $E(x) + \lambda Var(x)$ for our model and Almgren-Chriss model for each λ in Figure 5.12. We plot the relative error for the value of $E(x) + \lambda Var(x)$ for each test λ case. The error is shown as a percentage and is calculated as $\left| \frac{\text{our result} - \text{benchmark}}{\text{benchmark}} \right|$. The plot is shown in Figure 5.13. The results of our model align well with the benchmark results where the relative error remains within a range of 3%. As the λ value increases, this error tends to decrease. From Figures 5.12 and 5.13, we can conclude that the RL model provides a good estimate of $\min(E(x) + \lambda Var(x))$ when compared to the benchmark.

		1×10^{-6}	1.5×10^{-6}	2×10^{-6}
Our Result	stop period	25	22	23
	mean-variance value	725279.4062	821875.875	895509.875
Benchmark Result	stop period	54	43	39
	mean-variance value	704002.1875	803619.875	883295.

		2.5×10^{-6}	3×10^{-6}	3.5×10^{-6}
Our Result	stop period	21	21	23
	mean-variance value	964150.875	1018460.875	1061939.25
Benchmark Result	stop period	35	32	30
	mean-variance value	948965.875	1007413.25	1059865.125

Table 5.10: Comparison of Time Periods to Stop for Our Results and Optimal Results for different λ Values

Moreover, we compare the inventory list $\{q_t\}_{t=0}^N$, with $q_0 = Q$ and $q_N = 0$, for each λ values. The blue curves in Figure 5.14 represent our model's inventory list, while the orange curves correspond to the benchmark model's. The x-axis shows the time periods, and the y-axis indicates the number of shares held. From Figure 5.14, it is evident that the blue curves reach zero faster than the orange curves for all λ values tested. Additionally, the smaller the λ value, the quicker the blue curves tend to zero compared to the orange curves. Table 5.10 illustrates the stop periods for completing trades under various λ values. The data supports our observation: our model consistently finishes the trading process earlier than the benchmark model. For example, at $\lambda = 1 \times 10^{-6}$, our model completes trading in

25 periods, whereas the benchmark model takes 54 periods. This pattern holds across all λ values, with the gap in completion time widening as λ decreases. In summary, our RL model demonstrates superior efficiency by completing the liquidation process earlier across all λ values while maintaining comparable performance on the optimal trade execution problem. Finishing trades in fewer periods reduces exposure to market volatility and risk, making our model a more robust and preferable choice compared to the benchmark.

5.2.2 Nonlinear Price Impact Function

We follow most of the parameter settings in Table 5.9. The only difference lies in the temporary price impact function. We adopt the nonlinear temporary price impact function from [16], which has the following form:

$$h\left(\frac{u_t}{\tau}\right) = \left[1 + K_s \operatorname{sgn}\left(\frac{u_t}{\tau}\right)\right] \exp\left(K_t \operatorname{sgn}\left(\frac{u_t}{\tau}\right) \left|\frac{u_t}{\tau}\right|^v\right), \quad (5.2.2.1)$$

where $K_s = 2 \times 10^{-6}$, $K_t = 0$ and $v = 1$.

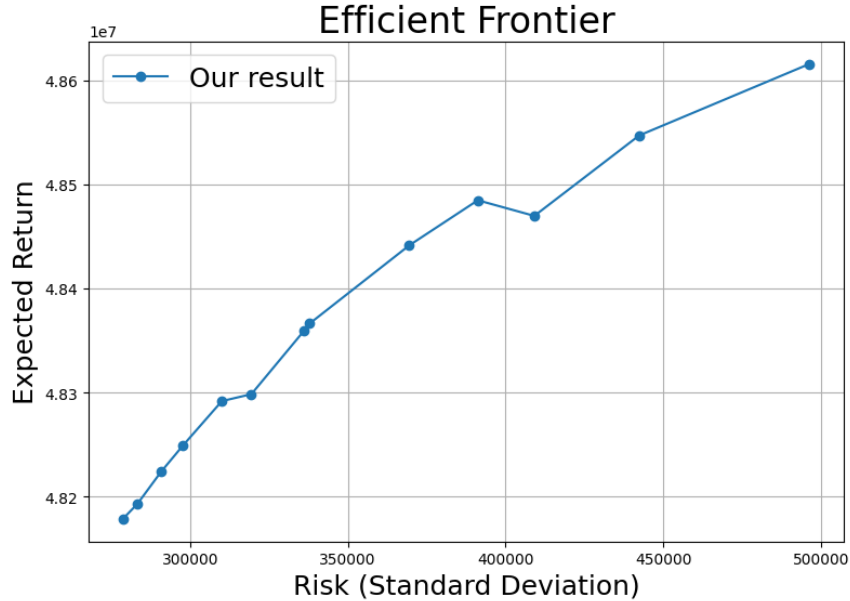


Figure 5.15: Efficient frontier curves for each λ value.

The efficient frontier plot for this case is shown in Figure 5.15. Overall, the curve demonstrates an expected trend where higher expected returns correspond to higher risk. However, there is one point that appears slightly abnormal, exhibiting a lower expected return compared to the nearest point to the left, yet with higher risk. Since the key evaluation metric is the combined value of $E(x) + \lambda Var(x)$, the optimal value can be achieved through different combinations of $E(x)$ and $Var(x)$. We have plotted the curve of $E(x) + \lambda Var(x)$ for our model in Figure 5.16. The trend of the minimum $E(x) + \lambda Var(x)$ appears accurate, reinforcing the notion that different combinations of $E(x)$ and $Var(x)$ can yield the same optimal $E(x) + \lambda Var(x)$ value.

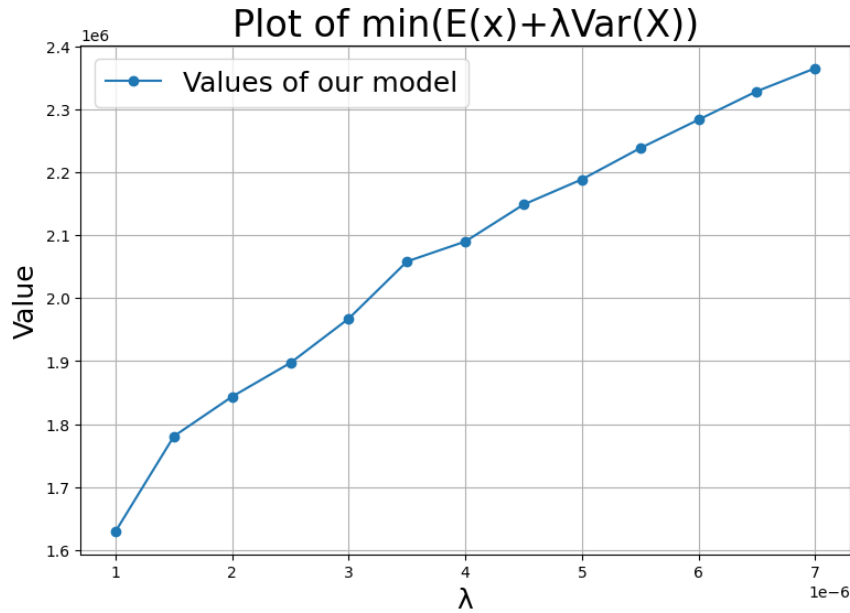


Figure 5.16: $\min(E(x) + \lambda \cdot \text{Var}(x))$ for each test λ case.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, we leverage reinforcement learning (RL) technology to address two key financial problems: risk management and optimal trade execution.

Building upon the work presented in [12], we integrate a dynamic Conditional Value at Risk (CVaR) risk measure to formulate a risk-averse RL model. Unlike traditional approaches that aim to maximize returns, our objective function minimizes cumulative costs, thus focusing on risk management. A notable improvement in our model is the reuse of policy network computations during value network evaluation, which significantly reduces computational costs and accelerates training time. Additionally, we optimize the trade-off between exploration and exploitation.

We demonstrate the robustness of our enhanced model through three distinct experiments, using the model from [12] as the baseline for comparison. First, under the same testing scenario as in [12], our model achieves superior results in both mean and CVaR values while reducing training time by approximately 50%. Further testing is conducted under scenarios involving market jumps and a traditional stock trading environment. In these cases, our model consistently outperforms the baseline in mean value and achieves better CVaR results in most cases. These results indicate that our enhancements contribute to a more efficient and robust RL model for risk management in portfolio optimization.

We also propose a novel RL framework for addressing the optimal trade execution problem, using the Almgren-Chriss model [3] as a benchmark. Both models are tested under a linear formulation of price evolution and price impact functions. Through extensive

computational experiments, we demonstrate that our RL model achieves an optimal mean-variance value with a relative error of less than 3% across 12 different values of λ . Moreover, our model consistently produces shorter trading lists compared to the benchmark, thereby completing trades more quickly and reducing exposure to market volatility and risk. This demonstrates that our RL model is a more robust and preferable choice compared to traditional methods.

Finally, we extend the application of our RL model to nonlinear conditions, where the price impact function is nonlinear, a setting in which the Almgren-Chriss model struggles to provide a closed-form solution. Even under these challenging conditions, our RL model exhibits the expected behavior, with efficient frontier plots showing the anticipated trend: higher expected returns are associated with higher risk. This demonstrates the flexibility and robustness of our RL framework, making it well-suited for more complex trading environments.

6.2 Future Work

This thesis explores utilizing reinforcement learning (RL) in financial applications, specifically within the domains of dynamic CVaR measurement and optimal trade execution. However, there are several avenues for future research that can further enhance these models.

First, as observed in Figure 5.2(c), both the baseline model and our enhanced RL framework struggled in convergence with extending to a longer time horizon ($N = 30$) under the presence of a terminal penalty. Since our framework builds upon the baseline, it is likely that limitations in the baseline model are being carried forward, causing both models to converge to local minima. Future research should focus on resolving this by refining the underlying algorithm, potentially introducing advanced optimization techniques or regularization to prevent such convergence issues and enable more robust performance over extended time horizons.

Second, our work has concentrated on the optimal trade execution problem only for a single stock scenario. The next step could be to extend this framework to multiple correlated assets, which would introduce more realistic, complex testing scenarios. Handling multiple assets would allow for the development of cross-asset strategies, managing asset correlations, and enhancing diversification—demonstrating the flexibility of RL in tackling scenarios beyond the reach of traditional methods.

Finally, variance remains a challenge in optimal trade execution, as calculating it requires multiple trajectories, making it computationally intensive to obtain a reliable mean-variance distribution. This complexity limits the direct application of risk measures, such as those presented in Chapter 3, to the optimal trade execution problem. Future work could aim to develop more efficient methods for managing variance, enabling a seamless integration of the risk-sensitive techniques discussed in Chapters 3 and 4. This would result in a more robust RL model capable of addressing both risk and execution challenges in financial markets.

References

- [1] Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.
- [2] Ammar Ahmed, Berman Kayis, and Sataporn Amornsawadwatana. A review of techniques for risk management in projects. *Benchmarking: an international journal*, 14(1):22–36, 2007.
- [3] Robert Almgren and Neil Chriss. Optimal execution of portfolio transactions. *Journal of Risk*, 3:5–40, 2001.
- [4] Saqib Aziz and Michael Dowling. Machine learning and ai for risk management. *Disrupting finance: FinTech and strategy in the 21st century*, pages 33–50, 2019.
- [5] Richard Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957.
- [6] Walden Bello, Nicola Bullard, and Kamal Malhotra. *Global Finance: New thinking on regulating speculative capital markets*. Zed Books, 2000.
- [7] Dimitri Bertsekas. Multiagent value iteration algorithms in dynamic programming and reinforcement learning. *Results in Control and Optimization*, 1:100003, 2020.
- [8] Dimitris Bertsimas and Andrew W Lo. Optimal control of execution costs. *Journal of financial markets*, 1(1):1–50, 1998.
- [9] Kang Boda and Jerzy A Filar. Time consistent dynamic risk measures. *Mathematical Methods of Operations Research*, 63:169–186, 2006.
- [10] John H Boyd, Jian Hu, and Ravi Jagannathan. The stock market’s reaction to unemployment news: Why bad news is usually good for stocks. *The Journal of Finance*, 60(2):649–672, 2005.

- [11] Yinlam Chow, Mohammad Ghavamzadeh, Lucas Janson, and Marco Pavone. Risk-constrained reinforcement learning with percentile risk criteria. *Journal of Machine Learning Research*, 18(167):1–51, 2018.
- [12] Anthony Coache and Sebastian Jaimungal. Reinforcement learning with dynamic convex risk measures. *Mathematical Finance*, 34(2):557–587, 2024.
- [13] Will Dabney, Mark Rowland, Marc Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.
- [14] Darrell Duffie and Jun Pan. An overview of value at risk. *Journal of derivatives*, 4(3):7–49, 1997.
- [15] Lorella Fatone, Francesca Mariani, Maria Cristina Recchioni, Francesco Zirilli, et al. A trading execution model based on mean field games and optimal control. *Applied Mathematics*, 5(19):3091, 2014.
- [16] Peter A Forsyth. A hamilton–jacobi–bellman approach to optimal trade execution. *Applied numerical mathematics*, 61(2):241–265, 2011.
- [17] Francesco Franzoni and Jose M Marin. Pension plan funding and stock market efficiency. *the Journal of Finance*, 61(2):921–956, 2006.
- [18] Alex Frino and Teddy Oetomo. Slippage in futures markets: Evidence from the sydney futures exchange. *Journal of Futures Markets: Futures, Options, and Other Derivative Products*, 25(12):1129–1146, 2005.
- [19] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.
- [20] Dieter Hendricks and Diane Wilcox. A reinforcement learning extension to the almgren-chriss framework for optimal trade execution. In *2014 IEEE Conference on computational intelligence for financial engineering & economics (CIFER)*, pages 457–464. IEEE, 2014.
- [21] Ronald Kahn and R Grinold. Active portfolio management. *A Quantitative Approach for Providing Superior Returns and Controlling Risk*, 1999.

- [22] Can B Kalayci, Okkes Ertenlice, and Mehmet Anil Akbay. A comprehensive review of deterministic models and applications for mean-variance portfolio optimization. *Expert Systems with Applications*, 125:345–368, 2019.
- [23] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [24] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.
- [25] Bjoern Krollner, Bruce Vanstone, and Gavin Finnie. Financial time series forecasting with machine learning techniques: A survey. In *European Symposium on Artificial Neural Networks: Computational Intelligence and Machine Learning*, pages 25–30, 2010.
- [26] Jaeyoung Lee and Richard S Sutton. Policy iterations for reinforcement learning problems in continuous time and space—fundamental theory and methods. *Automatica*, 126:109421, 2021.
- [27] Siyu Lin and Peter A Beling. An end-to-end optimal trade execution framework based on proximal policy optimization. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 4548–4554, 2021.
- [28] Jelena Luketina, Nantas Nardelli, Gregory Farquhar, Jakob Foerster, Jacob Andreas, Edward Grefenstette, Shimon Whiteson, and Tim Rocktäschel. A survey of reinforcement learning informed by natural language. *arXiv preprint arXiv:1906.03926*, 2019.
- [29] Ross A Maller, Gernot Müller, and Alex Szimayer. Ornstein–uhlenbeck processes and extensions. *Handbook of financial time series*, pages 421–437, 2009.
- [30] H. Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, mar 1952.
- [31] Harry M Markowitz and Erik Van Dijk. Risk-return analysis. In *Handbook of asset and liability management*, pages 139–197. Elsevier, 2008.
- [32] Alexander J McNeil, Rüdiger Frey, and Paul Embrechts. *Quantitative risk management: concepts, techniques and tools-revised edition*. Princeton university press, 2015.

- [33] Ishai Menache, Shie Mannor, and Nahum Shimkin. Basis function adaptation in temporal difference reinforcement learning. *Annals of Operations Research*, 134(1):215–238, 2005.
- [34] Nicholas Metropolis and Stanislaw Ulam. The monte carlo method. *Journal of the American statistical association*, 44(247):335–341, 1949.
- [35] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
- [36] Amy B Monahan. Public pension plan reform: The legal framework. *Education Finance and Policy*, 5(4):617–646, 2010.
- [37] Manuel Monge, María Fátima Romero Rojo, and Luis Alberiko Gil-Alana. The impact of geopolitical risk on the behavior of oil prices and freight rates. *Energy*, 269:126779, 2023.
- [38] R Tyrrell Rockafellar, Stanislav Uryasev, et al. Optimization of conditional value-at-risk. *Journal of risk*, 2:21–42, 2000.
- [39] Andrzej Ruszczyński. Risk-averse dynamic programming for markov decision processes. *Mathematical programming*, 125:235–261, 2010.
- [40] Andrzej Ruszczyński. Advances in risk-averse optimization. In *Theory driven by influential applications*, pages 168–190. INFORMS, 2013.
- [41] Andrzej Ruszczyński and Alexander Shapiro. Conditional risk mappings. *Mathematics of operations research*, 31(3):544–561, 2006.
- [42] Sergey Sarykalin, Gaia Serraino, and Stan Uryasev. Value-at-risk vs. conditional value-at-risk in risk management and optimization. In *State-of-the-art decision-making tools in the information-intensive age*, pages 270–294. Informs, 2008.
- [43] David Shale and W Forrest Stinespring. Wiener processes. *Journal of Functional Analysis*, 2(4):378–394, 1968.
- [44] Alexander Shapiro, Darinka Dentcheva, and Andrzej Ruszczyński. *Lectures on stochastic programming: modeling and theory*. SIAM, 2021.
- [45] René M Stulz. Optimal hedging policies. *Journal of Financial and Quantitative analysis*, 19(2):127–140, 1984.

- [46] René M Stulz. Rethinking risk management. In *Corporate risk management*, pages 87–120. Columbia University Press, 2008.
- [47] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- [48] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.
- [49] Gerald Tesauro et al. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.
- [50] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018(1):7068349, 2018.
- [51] Jiang Wang. A model of competitive stock trading volume. *Journal of political Economy*, 102(1):127–168, 1994.
- [52] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.
- [53] S Kevin Zhou, Hoang Ngan Le, Khoa Luu, Hien V Nguyen, and Nicholas Ayache. Deep reinforcement learning in medical imaging: A literature review. *Medical image analysis*, 73:102193, 2021.