

# Investigation To A Neural Network Approach To Optimal Dynamic Allocation Problem In Defined Contribution Pension Plans

by

Yuan Liu

A thesis

presented to the University of Waterloo

in fulfillment of the

thesis requirement for the degree of

Master of Mathematics

in

Data Science

Waterloo, Ontario, Canada, 2025

© Yuan Liu 2025

## Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the 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.

## Statement of Contributions

Yuan Liu was the sole author of Chapters 1, 5, and 6, which were written under the supervision of Drs. Peter Forsyth and Yuying Li.

Chapters 2, 3 and 4 consist of material from a manuscript that was written for publication. This manuscript was co-first authored by Marc Chen and Mohammad Shirazi and coauthored by Drs. Peter Forsyth and Yuying Li. The research was conducted at the University of Waterloo under the supervision of Drs. Peter Forsyth and Yuying Li. Marc Chen and Mohammad Shirazi were the primary coders for the framework presented in the manuscript.

Citation:

M. Chen, M. Shirazi, P. Forsyth, and Y. Li. “Machine Learning and Hamilton-JacobiBellman Equation for Optimal Decumulation: a Comparison Study.” (2023). ArXiv preprint. URL <https://arxiv.org/abs/2306.10582>.

# Abstract

In this thesis, we propose a data-driven neural network (NN) optimization framework for solving a dynamic stochastic control problem under stochastic constraints. The objective function of the optimal control problem is based on expected wealth withdrawn (EW) and expected shortfall (ES) that directly targets left-tail risk. The optimal solution obtained from NN framework achieves high computational accuracy comparable to the Hamilton-Jacobi-Bellman (HJB) Partial Differential Equation (PDE) method. Additionally, the NN framework exhibits strong computational robustness, maintaining stable performance across different data distributions. Unlike traditional HJB PDE approaches, the NN framework can be extendable to high-dimensional multi-asset problems, overcoming the curse of dimensionality. To further enhance data diversity and improve generalization, we introduce TimeGAN and incorporate TimeGAN-generated data to generate historical financial time-series data, ensuring the robustness of model training.

## Acknowledgment

I am deeply grateful to my advisors, Yuying Li and Peter Forsyth, for their guidance and support throughout this research. I also thank my family and friends for their unwavering encouragement and understanding. This achievement would not have been possible without their kindness and inspiration.

# Table of Contents

<b>Author’s Declaration</b>	<b>ii</b>
<b>Statement of Contributions</b>	<b>iii</b>
<b>Abstract</b>	<b>iv</b>
<b>Acknowledgment</b>	<b>v</b>
<b>List of Figures</b>	<b>viii</b>
<b>List of Tables</b>	<b>xv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Optimal Dynamic Withdrawal and Allocation Problem . . . . .	1
1.2 Existing Approaches . . . . .	1
1.3 Research Goal . . . . .	2
1.4 Thesis Outline . . . . .	3
<b>2 Problem Formulation</b>	<b>5</b>
2.1 Overview . . . . .	5
2.2 Notational Conventions . . . . .	5
2.3 Risk Measure: Expected Shortfall (ES) . . . . .	6
2.4 Reward Measure: Total Expected Withdrawals (EW) . . . . .	7
2.5 Objective Function . . . . .	7
<b>3 Data Generation</b>	<b>9</b>
3.1 Overview . . . . .	9
3.2 CRSP Historical Data . . . . .	9
3.3 Stochastic Model Calibration . . . . .	10
3.4 Bootstrap Resampling . . . . .	11
3.5 TimeGAN . . . . .	12
3.5.1 TimeGAN Framework . . . . .	12
3.5.2 CRSP TimeGAN Data Generation for Optimal Control Allocation Problem . . . . .	15
<b>4 Methods for Computing Optimal Dynamic Controls</b>	<b>18</b>
4.1 Overview . . . . .	18
4.2 HJB Dynamic Programming Optimization Framework . . . . .	18
4.2.1 Deriving Auxiliary Function for $PCEE_{t_0}(\kappa)$ . . . . .	18
4.2.2 Applying Dynamic Programming at Rebalancing Times . . . . .	19
4.2.3 Conditional Expectations between Rebalancing Times . . . . .	19
4.2.4 Equivalence with $PCEE_{t_0}(\kappa)$ . . . . .	20
4.3 Neural Network Formulation . . . . .	20
4.3.1 Neural Network Optimization for $PCEE_{t_0}(\kappa)$ . . . . .	21
4.3.2 Neural Network Framework . . . . .	21
4.3.3 NN Estimate of the Optimal Control . . . . .	23

4.3.4	Stochastic Gradient Method . . . . .	23
<b>5</b>	<b>Computational Investigation</b>	<b>27</b>
5.1	Investment Scenario and NN Parameter Settings . . . . .	27
5.2	Emphasis on Minimizing Risk . . . . .	28
5.2.1	Computational Challenges in Minimizing Risk . . . . .	28
5.2.2	Computational Results . . . . .	29
5.2.3	Transfer Learning . . . . .	31
5.3	Heatmap for Strategy Analysis . . . . .	39
5.3.1	HJB Control Heatmaps . . . . .	39
5.3.2	NN Control Heatmaps . . . . .	39
5.4	Updated Historical Data Training Results . . . . .	42
5.4.1	Asset Return . . . . .	42
5.4.2	Computational Results . . . . .	43
5.5	Multi-Assets Training Results . . . . .	47
5.6	Computational Investigation in TimeGAN Synthetic Data Generation . . . . .	55
5.6.1	TimeGAN Synthetic Data Generating Performance . . . . .	55
5.6.2	Investigation for TimeGAN Hyper-parameter Settings . . . . .	57
<b>6</b>	<b>Model Robustness</b>	<b>61</b>
6.1	CRSP Bootstrap Resampling Historical Data Testing . . . . .	61
6.1.1	Out-of-sample testing . . . . .	61
6.1.2	Out-of-distribution testing . . . . .	62
6.1.3	Sensitivity to training distribution . . . . .	63
6.2	TimeGAN Data Testing . . . . .	64
<b>7</b>	<b>Conclusion</b>	<b>67</b>
	<b>References</b>	<b>69</b>

# List of Figures

3.1	<i>TimeGAN Framework Training Process. (a) Block diagram of component functions and objectives. (b) Training scheme; solid lines indicate forward propagation of data, and dashed lines indicate backpropagation of gradients. (Yoon et al., 2019)</i>	15
4.1	<i>Illustration of the NN framework as per Section 4.3. (Chen et al., 2023)</i>	22
5.1	<i>Comparison of direct training results for the <math>\kappa = 0.05</math> model and <math>\kappa = 50</math> model by Neural Network (NN) method, with controls computed from Problem (2.18). Investment setup in Table 5.1. Hyper-parameters setup in Table 5.2. Model was trained on <math>2.56 \times 10^5</math> observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12.</i>	30
5.2	<i>Comparison of (EW, ES) outcomes for training <math>\kappa = 0.2</math> model, with controls computed from Problem (2.18). The <math>\kappa = 0.05</math> model is used as the source model for transfer learning to <math>\kappa = 0.2</math> models. Model was trained on <math>2.56 \times 10^5</math> observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12. The investment setup is detailed in Table 5.1, and the hyper-parameter configuration is provided in Table 5.2.</i>	33
5.3	<i>Comparison of (EW, ES) outcomes for training <math>\kappa = 0.5</math> model, with controls computed from Problem (2.18). The <math>\kappa = 0.05</math> model is used as the source model for transfer learning to <math>\kappa = 0.5</math> models. Model was trained on <math>2.56 \times 10^5</math> observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12. The investment setup is detailed in Table 5.1, and the hyper-parameter configuration is provided in Table 5.2.</i>	33
5.4	<i>Comparison of (EW, ES) outcomes for training <math>\kappa = 1.0</math> model, with controls computed from Problem (2.18). The <math>\kappa = 0.05</math> model is used as the source model for transfer learning to <math>\kappa = 1.0</math> models. Model was trained on <math>2.56 \times 10^5</math> observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12. The investment setup is detailed in Table 5.1, and the hyper-parameter configuration is provided in Table 5.2.</i>	34
5.5	<i>Comparison of (EW, ES) outcomes for training <math>\kappa = 1.5</math> model, with controls computed from Problem (2.18). The <math>\kappa = 0.05</math> model is used as the source model for transfer learning to <math>\kappa = 1.5</math> models. Model was trained on <math>2.56 \times 10^5</math> observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12. The investment setup is detailed in Table 5.1, and the hyper-parameter configuration is provided in Table 5.2.</i>	34

5.6	Comparison of (EW, ES) outcomes for training $\kappa = 3.0$ model, with controls computed from Problem (2.18). The $\kappa = 0.05$ model is used as the source model for transfer learning to $\kappa = 3.0$ models. Model was trained on $2.56 \times 10^5$ observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12. The investment setup is detailed in Table 5.1, and the hyper-parameter configuration is provided in Table 5.2. . . . .	35
5.7	Comparison of (EW, ES) outcomes for training $\kappa = 5.0$ model, with controls computed from Problem (2.18). The $\kappa = 0.05$ model is used as the source model for transfer learning to $\kappa = 5.0$ models. Model was trained on $2.56 \times 10^5$ observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12. The investment setup is detailed in Table 5.1, and the hyper-parameter configuration is provided in Table 5.2. . . . .	35
5.8	Comparison of (EW, ES) outcomes for training $\kappa = 50.0$ model, with controls computed from Problem (2.18). The $\kappa = 0.05$ model is used as the source model for transfer learning to $\kappa = 50.0$ models. Model was trained on $2.56 \times 10^5$ observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12. The investment setup is detailed in Table 5.1, and the hyper-parameter configuration is provided in Table 5.2. . . . .	36
5.9	Comparison of (EW, ES) efficient frontiers for training performances between sequential transfer learning starting from the $\kappa = 0.05$ model and no transfer learning by Neural Network (NN) method. Model was trained on $2.56 \times 10^5$ observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12. Labels on notes indicate $\kappa$ parameter values for NN results. The investment setup is detailed in Table 5.1, and the hyper-parameter configuration is provided in Table 5.2. . . . .	37
5.10	(EW, ES) efficient frontier of training performance for sequential transfer learning starting from the $\kappa = 1.0$ model by Neural Network (NN) method. Labels on notes indicate $\kappa$ parameter values for NN results. Model was trained on $2.56 \times 10^5$ observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12. The investment setup is detailed in Table 5.1, and the hyper-parameter configuration is provided in Table 5.2. . . . .	38
5.11	Fraction in stocks heatmap and wealth percentiles with controls computed from Problem (2.18). Note: investment setup is as in Table 5.1. HJB solution performance computed on $2.56 \times 10^6$ observations of simulated data from calibrated jump models. Parameters for simulated data based on CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD), from 1926:1 to 2019:12, are shown in Table 3.1. Minimum withdrawal: 35. Maximum withdrawal: 60. The wealth percentiles are calculated based on post-withdrawal wealth $W_t^+$ , while the asset allocation heatmaps are computed from post-withdrawal allocation decisions. $\kappa = 1.0$ . $W_0 = 58.0$ for PIDE results. (a) $\epsilon = 10^{-6}$ . (b) $\epsilon = -10^{-6}$ . Monetary units: USD\$ in thousands. . . . .	39
5.12	Fraction in stocks heatmap and wealth percentiles with controls computed from Problem (2.18). Note: investment setup is as in Table 5.1. Hyper-parameters setup in Table 5.2. NN solution performance trained on $2.56 \times 10^5$ observations of simulated data from calibrated jump models. Parameters for simulated data based on CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD), from 1926:1 to 2019:12, are shown in Table 3.1. Minimum withdrawal: 35. Maximum withdrawal: 60. The wealth percentiles are calculated based on pre-withdrawal wealth $W_t^-$ , while the asset allocation heatmaps are computed from post-withdrawal allocation decisions. $\kappa = 1.0$ . Monetary units: USD\$ in thousands. . . . .	41

5.13	<i>Fraction in stocks heatmap and wealth percentiles with controls computed from Problem (2.18). Note: investment setup is as in Table 5.1. Hyper-parameters setup in Table 5.2. NN solution performance trained on <math>2.56 \times 10^5</math> observations of simulated data from calibrated jump models. Parameters for simulated data based on CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD), from 1926:1 to 2019:12, are shown in Table 3.1. Minimum withdrawal: 35. Maximum withdrawal: 60. The wealth percentiles are calculated based on pre-withdrawal wealth <math>W_t^-</math>, while the asset allocation heatmaps are computed from post-withdrawal allocation decisions. <math>\kappa = 1.0</math>. Monetary units: USD\$ in thousands.</i>	42
5.14	<i>Comparison of (EW, ES) efficient frontiers for NN training performance on CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) over two different time spans: 1926:1 - 2019:12 and 1926:1 - 2023:12. Controls are computed from Problem (2.18). Model trained on <math>2.56 \times 10^5</math> observations of resampled data from bootstrap resampling. Investment setup is detailed in Table 5.1, and hyper-parameter configurations are provided in Table 5.2. Expected block size of bootstrap resampling is 3, see Table 3.2.</i>	45
5.15	<i>Comparison of the cumulative distribution functions (CDF) of 30-year log returns for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) across different time spans. The datasets are generated using <math>2.56 \times 10^5</math> observations of bootstrap resampling data from CRSP 1926:1–2019:12, CRSP 1926:1–2023:12, and CRSP 2020:1–2023:12. Expected block size of bootstrap resampling is 3, see Table 3.2.</i>	45
5.16	<i>B10 allocation percentiles for NN training performance on CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) over 1926:1 - 2023:12. Controls are computed from Problem (2.18). Model trained on <math>2.56 \times 10^5</math> observations of resampled data from bootstrap resampling. Investment setup is detailed in Table 5.1, and hyper-parameter configurations are provided in Table 5.2. Expected block size of bootstrap resampling is 3, see Table 3.2.</i>	46
5.17	<i>Fraction in stocks heatmap and wealth percentiles with controls computed from Problem (2.18). Note: investment setup is as in Table 5.1. Hyper-parameters setup in Table 5.2. NN solution performance computed on <math>2.56 \times 10^5</math> observations of bootstrap resampling CPI adjusted CRSP historical data (CRSP 10-Year T-Bond Returns (B10) &amp; CRSP Value-Weighted Index (VWD)) from 1926:1 to 2023:12. The wealth percentiles are calculated based on pre-withdrawal wealth <math>W_t^-</math>, while the asset allocation heatmaps are computed from post-withdrawal allocation decisions. Expected block size of bootstrap resampling is 3, see Table 3.2. <math>\kappa = 1.0</math>. Monetary units: USD\$ in thousands.</i>	47
5.18	<i>Comparison of the cumulative distribution functions (CDF) of 30-year log returns for CPI adjusted CRSP 30-Day T-Bill Returns (T30), CRSP 90-Day T-Bill Returns (T90), and CRSP 10-Year T-Bond Returns (B10), from 1926:1 to 2023:12. The datasets are generated using <math>2.56 \times 10^5</math> observations of bootstrap resampled data. Expected block size of bootstrap resampling is 3, see Table 3.2.</i>	48
5.19	<i>Comparison of (EW, ES) efficient frontiers for the Neural Network (NN) training for different CPI adjusted CRSP bond assets : CRSP 10-Year T-Bond Returns (B10) &amp; CRSP 30-Day T-Bill Returns (T30) &amp; CRSP 90-Day T-Bill Returns (T90), together with CPI adjusted CRSP Value-Weighted Index (VWD), computed from the Problem (2.18). Note: investment setup in Table 5.1. Hyper-parameters setup in Table 5.2. Control computed from the NN model, trained on <math>2.56 \times 10^5</math> observations of bootstrapping resampling CRSP historical data for each bond asset from 1926:1 to 2023:12. Expected block size of bootstrap resampling is 3, see Table 3.2.</i>	49

5.20	<i>Fraction in stocks heatmap and wealth percentiles, and normalized withdrawal with controls computed from Problem (2.18). Note: investment setup is as in Table 5.1. Hyper-parameters setup in Table 5.2. NN solution performance computed on <math>2.56 \times 10^5</math> observations of bootstrap resampling CPI adjusted CRSP historical data (CRSP 10-Year T-Bond Returns (B10) &amp; CRSP Value-Weighted Index (VWD)) from 1926:1 to 2023:12. The wealth percentiles are calculated based on pre-withdrawal wealth <math>W_t^-</math>, while the asset allocation heatmaps are computed from post-withdrawal allocation decisions. Expected block size of bootstrap resampling is 3, see Table 3.2. <math>\kappa = 1.0</math>. Monetary units: USD\$ in thousands. . . . .</i>	49
5.21	<i>Fraction in stocks heatmap and wealth percentiles, and normalized withdrawal, with controls computed from Problem (2.18). Note: investment setup is as in Table 5.1. Hyper-parameters setup in Table 5.2. NN solution performance computed on <math>2.56 \times 10^5</math> observations of bootstrap resampling CPI adjusted CRSP historical data (CRSP 30-Day T-Bill Returns (T30) &amp; CRSP Value-Weighted Index (VWD)) from 1926:1 to 2023:12. The wealth percentiles are calculated based on pre-withdrawal wealth <math>W_t^-</math>, while the asset allocation heatmaps are computed from post-withdrawal allocation decisions. Expected block size of bootstrap resampling is 3, see Table 3.2. Expected block size of bootstrap resampling is 3, see Table 3.2. <math>\kappa = 1.0</math>. Monetary units: USD\$ in thousands. . . . .</i>	50
5.22	<i>Fraction in stocks heatmap and wealth percentiles, and normalized withdrawal, with controls computed from Problem (2.18). Note: investment setup is as in Table 5.1. Hyper-parameters setup in Table 5.2. NN solution performance computed on <math>2.56 \times 10^5</math> observations of bootstrap resampling CPI adjusted CRSP historical data (CRSP 90-Day T-Bill Returns (T90) &amp; CRSP Value-Weighted Index (VWD)) from 1926:1 to 2023:12. The wealth percentiles are calculated based on pre-withdrawal wealth <math>W_t^-</math>, while the asset allocation heatmaps are computed from post-withdrawal allocation decisions. Expected block size of bootstrap resampling is 3, see Table 3.2. <math>\kappa = 1.0</math>. Monetary units: USD\$ in thousands. . . . .</i>	50
5.23	<i>Comparison of the cumulative distribution functions (CDF) of 30-year log returns for CPI adjusted CRSP Value-Weighted Index (VWD), and CRSP Equal-Weighted Index (EWD), from 1926:1 to 2023:12. The datasets are generated from <math>2.56 \times 10^5</math> bootstrap resampled data. Expected block size of bootstrap resampling is 3, see Table 3.2. . . . .</i>	51
5.24	<i>Comparison of (EW, ES) efficient frontiers for the Neural Network (NN) training for different CPI adjusted CRSP equity assets : CRSP Value-Weighted Index (VWD) &amp; CRSP Equal-Weighted Index (EWD), together with CPI adjusted CRSP 10-Year T-Bond Returns (B10), computed from the Problem (2.18). Note: investment setup in Table 5.1. Hyper-parameters setup in Table 5.2. Control computed from the NN model, trained on <math>2.56 \times 10^5</math> observations of bootstrapping resampling CRSP historical data for each equity asset from 1926:1 to 2023:12. Expected block size of bootstrap resampling is 3, see Table 3.2. . . . .</i>	52
5.25	<i>Fraction in stocks heatmap and wealth percentiles, and normalized withdrawal, with controls computed from Problem (2.18). Note: investment setup is as in Table 5.1. Hyper-parameters setup in Table 5.2. NN solution performance computed on <math>2.56 \times 10^5</math> observations of bootstrap resampling CPI adjusted CRSP historical data (CRSP 10-Year T-Bond Returns (B10) &amp; CRSP Value-Weighted Index (VWD)) from 1926:1 to 2023:12. The wealth percentiles are calculated based on pre-withdrawal wealth <math>W_t^-</math>, while the asset allocation heatmaps are computed from post-withdrawal allocation decisions. Expected block size of bootstrap resampling is 3, see Table 3.2. <math>\kappa = 1.0</math>. Monetary units: USD\$ in thousands. . . . .</i>	53

5.26	<i>Fraction in stocks heatmap and wealth percentiles, and normalized withdrawal with controls computed from Problem (2.18). Note: investment setup is as in Table 5.1. Hyper-parameters setup in Table 5.2. NN solution performance computed on <math>2.56 \times 10^5</math> observations of bootstrap resampling CPI adjusted CRSP historical data (CRSP 10-Year T-Bond Returns (B10) &amp; CRSP Equal-Weighted Index (EWD)) from 1926:1 to 2023:12. The wealth percentiles are calculated based on pre-withdrawal wealth <math>W_t^-</math>, while the asset allocation heatmaps are computed from post-withdrawal allocation decisions. Expected block size of bootstrap resampling is 3, see Table 3.2. <math>\kappa = 1.0</math>. Monetary units: USD\$ in thousands. . . . .</i>	53
5.27	<i>Fraction in assets heatmaps and wealth percentiles, and normalized withdrawal, with controls computed from Problem (2.18). Note: investment setup is as in Table 5.1. Hyper-parameters setup in Table 5.2. NN solution performance computed on <math>2.56 \times 10^5</math> observations of bootstrap resampling CPI adjusted CRSP historical data (CRSP 10-Year T-Bond Returns (B10) &amp; CRSP 30-Day T-Bill Returns (T30) &amp; CRSP 90-Day T-Bill Returns (T90) &amp; CRSP Value-Weighted Index (VWD) &amp; CRSP Equal-Weighted Index (EWD)) from 1926:1 to 2023:12. The wealth percentiles are calculated based on pre-withdrawal wealth <math>W_t^-</math>, while the asset allocation heatmaps are computed from post-withdrawal allocation decisions. Expected block size of bootstrap resampling is 3, see Table 3.2. <math>\kappa = 1.0</math>. Monetary units: USD\$ in thousands. . . . .</i>	54
5.28	<i>Comparison of (EW, ES) efficient frontiers for the Neural Network (NN) training for different CRSP multi-assets combinations: i) CRSP 10-Year T-Bond Returns (B10) &amp; CRSP Equal-Weighted Index (EWD), ii) CRSP 10-Year T-Bond Returns (B10) &amp; CRSP 90-Day T-Bill Returns (T90) &amp; CRSP Equal-Weighted Index (EWD), iii) CRSP 10-Year T-Bond Returns (B10) &amp; CRSP 30-Day T-Bill Returns (T30) &amp; CRSP 90-Day T-Bill Returns (T90) &amp; CRSP Value-Weighted Index (VWD) &amp; CRSP Equal-Weighted Index (EWD), computed from Problem (2.18). Note: investment setup in Table 5.1. Hyper-parameters setup in Table 5.2. Control computed from the NN model, trained on <math>2.56 \times 10^5</math> observations of bootstrapping resampling CPI adjusted CRSP historical data from 1926:1 to 2023:12. Expected block size of bootstrap resampling is 3, see Table 3.2. . . . .</i>	55
5.29	<i>Comparison of the cumulative distribution functions (CDF) of 30-year log returns between the bootstrapping resampling historical data and the TimeGAN-generated historical data (CPI adjusted CRSP 10-Year T-Bond Returns (B10) &amp; CRSP Value-Weighted Index (VWD), from 1926:1 to 2023:12). The data sets are generated using <math>2.56 \times 10^5</math> observations of data for each curve. Expected block size of bootstrap resampling is 3, see Table 3.2. TimeGAN hyper-parameter settings are presented in Table 3.4, with varying <math>(\lambda, \eta)</math> combinations. . . . .</i>	56
5.30	<i>Comparison of the cumulative distribution functions (CDF) of 30-year log returns between the bootstrapping resampling historical data and the TimeGAN-generated historical data (CPI adjusted CRSP 10-Year T-Bond Returns (B10), from 1926:1 to 2023:12). The data sets are generated using <math>2.56 \times 10^5</math> observations of data for each curve. Expected block size of bootstrap resampling is 3, see Table 3.2. TimeGAN hyper-parameter settings are presented in Table 3.4, with varying <math>(\lambda, \eta)</math> combinations. 1:(0.1,100), 2:(0.5,50), 3:(1,10), 4:(10,1), 5:(50,0.5), 6:(100,0.1) . . .</i>	58
5.31	<i>Comparison of the cumulative distribution functions (CDF) of 30-year log returns between the bootstrapping resampling historical data and the TimeGAN-generated historical data (CPI adjusted CRSP Value-Weighted Index (VWD), from 1926:1 to 2023:12). The data sets are generated using <math>2.56 \times 10^5</math> observations of data for each curve. Expected block size of bootstrap resampling is 3, see Table 3.2. TimeGAN hyper-parameter settings are presented in Table 3.4, with varying <math>(\lambda, \eta)</math> combinations. 1:(0.1,100), 2:(0.5,50), 3:(1,10), 4:(10,1), 5:(50,0.5), 6:(100,0.1) . . .</i>	58

5.32	<i>Comparison of the cumulative distribution functions (CDF) of 30-year log returns between the bootstrapping resampling historical data and the TimeGAN-generated historical data (CPI adjusted CRSP 10-Year T-Bond Returns (B10), from 1926:1 to 2023:12). The data sets are generated using <math>2.56 \times 10^5</math> observations of data for each curve. Expected block size of bootstrap resampling is 3, see Table 3.2. TimeGAN hyper-parameter settings are presented in Table 3.4, with varying sequence lengths for 3 months, and 6 months.</i>	59
5.33	<i>Comparison of the cumulative distribution functions (CDF) of 30-year log returns between the bootstrapping resampling historical data and the TimeGAN-generated historical data (CPI adjusted CRSP Value-Weighted Index (VWD), from 1926:1 to 2023:12). The data sets are generated using <math>2.56 \times 10^5</math> observations of data for each curve. Expected block size of bootstrap resampling is 3, see Table 3.2. TimeGAN hyper-parameter settings are presented in Table 3.4, with varying sequence lengths for 24 months, and 36 months.</i>	59
6.1	<i>Out-of-sample test. Comparison of NN training performance results vs. out-of-sample test using (EW, ES) efficient frontiers. Both training and testing datasets are <math>2.56 \times 10^5</math> observations of new historical data (1926:1 - 2023:12), generated with different random seeds. Note: investment setup in Table 5.1. Hyper-parameters setup in Table 5.2.</i>	62
6.2	<i>Out-of-distribution test. Comparison of NN training performance results vs. out-of-distribution test using (EW, ES) efficient frontier. The training (EW, ES) efficient frontier is on the new historical dataset (1926:1 - 2023:12), with testing (EW, ES) efficient frontier on model trained on old historical data (1926:1 - 2019:12) on new data. Both training and testing data sets contain <math>2.56 \times 10^5</math> observations. Note: investment setup in Table 5.1. Hyper-parameters setup in Table 5.2.</i>	62
6.3	<i>Out-of-distribution test. Comparison of NN training performance results vs. out-of-distribution test using (EW, ES) efficient frontier. The training (EW, ES) efficient frontier is on the old historical dataset (1926:1 - 2019:12), with testing (EW, ES) efficient frontier on model trained on new historical data (1926:1 - 2023:12) on old data. Both training and testing data sets contain <math>2.56 \times 10^5</math> observations. Note: investment setup in Table 5.1. Hyper-parameters setup in Table 5.2.</i>	63
6.4	<i>Training on new historical data. (EW, ES) efficient frontiers of controls generated by NN model trained on <math>2.56 \times 10^5</math> observations of new historical data (1926:1 - 2023:12) with expected block sizes of a) 1 month and b) 12 months, each tested on <math>2.56 \times 10^5</math> observations of new historical data with expected block sizes of 3 months. Note: investment setup in Table 5.1. Hyper-parameters setup in Table 5.2.</i>	64
6.5	<i>(EW, ES) efficient frontiers of controls generated by NN model trained on <math>2.56 \times 10^5</math> observations of TimeGAN data, tested on <math>2.56 \times 10^5</math> observations of bootstrapping resampling historical data from 1926:1 to 2023:12. Note: investment setup in Table 5.1. Hyper-parameters setup in Table 5.2.</i>	65
6.6	<i>Fraction in stocks heatmap and wealth percentiles generated by NN model trained on <math>2.56 \times 10^5</math> observations of TimeGAN data, tested on <math>2.56 \times 10^5</math> observations of bootstrapping resampling historical data from 1926:1 to 2023:12. Note: investment setup in Table 5.1. Hyper-parameters setup in Table 5.2. The wealth percentiles are calculated based on pre-withdrawal wealth <math>W_t^-</math>, while the asset allocation heatmaps are computed from post-withdrawal allocation decisions. <math>\kappa = 1.0</math>. Monetary units: USD\$ in thousands.</i>	65
6.7	<i>(EW, ES) efficient frontiers of controls generated by NN model trained on <math>2.56 \times 10^5</math> observations of bootstrapping resampling historical data from 1926:1 to 2023:12, tested on <math>2.56 \times 10^5</math> observations of TimeGAN data. Note: investment setup in Table 5.1. Hyper-parameters setup in Table 5.2.</i>	66

6.8	<i>Fraction in stocks heatmap and wealth percentiles generated by NN model trained on <math>2.56 \times 10^5</math> observations of bootstrapping resampling historical data from 1926:1 to 2023:12, tested on <math>2.56 \times 10^5</math> observations of TimeGAN data. Note: investment setup in Table 5.1. Hyper-parameters setup in Table 5.2. The wealth percentiles are calculated based on pre-withdrawal wealth <math>W_t^-</math>, while the asset allocation heatmaps are computed from post-withdrawal allocation decisions. <math>\kappa = 1.0</math>. Monetary units: USD\$ in thousands.</i>	66
-----	--	----

# List of Tables

3.1	<i>Calibrated (annualized) parameters for double exponential jump diffusion model. CPI adjusted CRSP US Total Market Index and CRSP US 10-year treasury, also inflation adjusted. Data from 1926:1 to 2019:12.</i>	11
3.2	<i>Optimal expected blocksize <math>\hat{b} = 1/\nu</math>, from Patton et al. (2009). The blocksize is a draw from a geometric distribution with <math>\Pr(b = k) = (1 - \nu)^{k-1}\nu</math>.</i>	12
3.3	<i>Notation and Description of Variables in TimeGAN: An overview of the key symbols used in the TimeGAN framework, detailing their roles in capturing temporal dependencies and generating synthetic time-series data. It includes representations of real and synthetic features, latent embeddings, trainable parameters for each network component, and the discriminator’s decision variables.</i>	13
3.4	<i>TimeGAN Framework Training Hyper-parameter Settings for CPI Adjusted CRSP Data (CRSP 10-Year T-Bond Returns &amp; CRSP Value-Weighted Index, from 1926:1 to 2023:12) in Optimal Control Allocation Problem.</i>	17
5.1	<i>Problem setup and input data. Monetary units: USD\$ in thousands.</i>	27
5.2	<i>Hyper-parameters used in training the NN framework for numerical experiments. (Chen et al., 2023)</i>	28
5.3	<i>Comparison of direct training results for the <math>\kappa = 0.05</math> model and <math>\kappa = 50</math> model by Neural Network (NN) method, with controls computed from Problem (2.18). Investment setup in Table 5.1. Hyper-parameters setup in Table 5.2 with varying learning rate decay schedules. Model was trained on <math>2.56 \times 10^5</math> observations of simulated data from calibrated jump models.</i>	29
5.4	<i>Comparison of direct training results for the <math>\kappa = 50</math> model by Neural Network (NN) method with varying hyper-parameter settings, with controls computed from Problem (2.18). Investment setup in Table 5.1. Hyper-parameters setup in Table 5.2 with decreasing initial learning rates. Model was trained on <math>2.56 \times 10^5</math> observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12.</i>	30
5.5	<i>Comparison of direct training results for the <math>\kappa = 50</math> model by Neural Network (NN) method with varying hyper-parameter settings, with controls computed from Problem (2.18). Investment setup in Table 5.1. Hyper-parameters setup in Table 5.2 with increasing batch size. Model was trained on <math>2.56 \times 10^5</math> observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12.</i>	31
5.6	<i>Separate single transfer learning results for training each <math>\kappa</math> model, with controls computed from Problem (2.18). The <math>\kappa = 0.05</math> model is used as the source model for transfer learning to other <math>\kappa</math> models. Model was trained on <math>2.56 \times 10^5</math> observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12. The investment setup is detailed in Table 5.1, and the hyper-parameter configuration is provided in Table 5.2. Each model is trained five times with different random seeds (0, 1, 2, 3, 4) to examine consistency across runs, refer to Section 5.1.</i>	32

5.7	<i>The objective function values obtained from training with sequential transfer learning starting from <math>\kappa = 1.0</math> model, with controls computed from Problem (2.18). Model was trained on <math>2.56 \times 10^5</math> observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12. Investment setup in Table 5.1. Hyper-parameters setup in Table 5.2. Each model is trained five times with different random seeds (0, 1, 2, 3, 4) to examine consistency across runs, refer to Section 5.1.</i>	37
5.8	<i>The averaged expected withdrawal and expected shortfall obtained from training with sequential transfer learning starting from <math>\kappa = 1.0</math> model, with controls computed from Problem (2.18). Model was trained on <math>2.56 \times 10^5</math> observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12. Investment setup in Table 5.1. Hyper-parameters setup in Table 5.2. Each model is trained five times with different random seeds (0, 1, 2, 3, 4) to examine consistency across runs, refer to Section 5.1.</i>	38
5.9	<i>Objective function values for NN training performance on CPI adjusted CRSP 10-Year T-Bond Returns (B10) &amp; CRSP Value-Weighted Index (VWD), from 1926:1 to 2023:12, with controls computed from Problem (2.18). Model trained on <math>2.56 \times 10^5</math> observations of resampled data from bootstrap resampling. Investment setup in Table 5.1. Hyper-parameters setup in Table 5.2. Expected block size of bootstrap resampling is 3, see Table 3.2. Each model is trained five times with different random seeds (0, 1, 2, 3, 4) to examine consistency across runs, refer to Section 5.1.</i>	43
5.10	<i>Expected shortfall for NN training performance on CPI adjusted CRSP 10-Year T-Bond Returns (B10) &amp; CRSP Value-Weighted Index (VWD), from 1926:1 to 2023:12, with controls computed from Problem (2.18). Model trained on <math>2.56 \times 10^5</math> observations of resampled data from bootstrap resampling. Investment setup in Table 5.1. Hyper-parameters setup in Table 5.2. Expected block size of bootstrap resampling is 3, see Table 3.2. Each model is trained five times with different random seeds (0, 1, 2, 3, 4) to examine consistency across runs, refer to Section 5.1.</i>	44
5.11	<i>Expected withdrawal for NN training performance on CPI adjusted CRSP 10-Year T-Bond Returns (B10) &amp; CRSP Value-Weighted Index (VWD), from 1926:1 to 2023:12, with controls computed from Problem (2.18). Model trained on <math>2.56 \times 10^5</math> observations of resampled data from bootstrap resampling. Investment setup in Table 5.1. Hyper-parameters setup in Table 5.2. Expected block size of bootstrap resampling is 3, see Table 3.2. Each model is trained five times with different random seeds (0, 1, 2, 3, 4) to examine consistency across runs, refer to Section 5.1.</i>	44

# Chapter 1

## Introduction

### 1.1 Optimal Dynamic Withdrawal and Allocation Problem

A Defined Contribution (DC) pension plan is a retirement savings plan in which individuals contribute a portion of their income to investment accounts. Unlike Defined Benefit (DB) plans, where retirees receive a predetermined payout, DC plans shift the responsibility of investment decisions and risk management to individual investors, which Nobel Laureate William Sharpe referred to as “the nastiest, hardest problem in finance” (Ritholz, 2017). This shift makes withdrawal and asset allocation strategies crucial for ensuring financial stability during retirement. The withdrawal strategy determines the amount of wealth withdrawn at each period to sustain individual consumption, while the allocation strategy dictates how the remaining wealth is distributed among available investment options to optimize long-term rewards and risk management. However, as each individual’s financial situation differs, a uniform withdrawal and investment policy may not be suitable for everyone (Bernhardt and Donnelly, 2018; Bender et al., 2013). Consequently, developing personalized withdrawal and investment strategies has become increasingly important, especially in the face of economic uncertainty.

Determining how to withdraw and allocate investments across multiple assets is a topic of significant concern for many investors. Given the various uncertainties such as asset price fluctuations, market conditions, investors’ differing preferences, and manipulative practices, static investment strategies are not optimal for maximizing rewards and minimizing risk. Consequently, dynamically adjusting investment strategies in response to market trends can better adapt to the complexities of the financial market. The Optimal Dynamic Allocation Problem typically requires making adjustments to withdrawal and allocation strategies at different time points to achieve the optimization of the objective function. The objective function often involves goals such as maximizing rewards and minimizing risks. The Optimal Dynamic Allocation Problem holds significant importance in various domains, including portfolio management and pension fund management. An effective dynamic investment strategy can help investors adapt to changing market conditions, maximize rewards, and mitigate risks simultaneously.

### 1.2 Existing Approaches

Existing approaches to addressing the challenges of decumulation strategies in defined contribution (DC) pension plans have evolved over time. Traditional methods, such as the Bengen Rule (4% Rule), recommend withdrawing a constant 4% (adjusted for inflation) of initial capital each year and allocating investments equally between stocks and bonds (Bengen, 1994). While widely adopted, this strategy has been acknowledged to be sub-optimal (Williams and Kawashima, 2023).

More advanced techniques, such as the Hamilton-Jacobi-Bellman (HJB) Partial Differential Equation (PDE) framework, formulate the decumulation problem as a stochastic optimal control problem, solved using dynamic programming (Forsyth, 2022). The HJB PDE method models wealth evolution over a 30-year investment horizon. This method seeks to maximize expected withdrawals while minimizing the probability of depleting savings, balancing these conflicting

objectives through Pareto optimal solutions. To ensure retirees receive a minimum guaranteed income, a lower bound on withdrawals is imposed, while an upper bound defines a target withdrawal level. Additionally, the model enforces investment constraints, such as no shorting and no leverage. The resulting dynamic stochastic strategy is designed to adapt to market conditions and investor preferences, consistent with the principle of *planning to live, not planning to die* (Pfau, 2018).

However, despite its theoretical rigor, the PDE method suffers from high computational complexity, making it impractical for high-dimensional problems involving multiple assets. The curse of dimensionality significantly limits its scalability, requiring approximations or alternative methods for real-world applications. Additionally, the PDE approach relies on a parametric model, which can introduce model misspecification risks and reduce flexibility in computing from the data sets directly. These challenges have motivated the development of neural network-based approaches, which offer greater flexibility, computational efficiency, and adaptability in solving decumulation and allocation problems.

Various neural network methods have been proposed for stochastic optimal control in finance. Buehler et al. (2019) introduced a stacked NN model for deep hedging, while Han and E. (2016) applied deep learning to high-dimensional control problems. Huré et al. (2021) combined dynamic programming with reinforcement learning. Tsang and Wong (2020) leveraged deep learning for portfolio selection with serially dependent returns. Li and Forsyth (2019) proposed a data-driven neural network framework for optimal asset allocation in target-based defined contribution pension plans. These methods present the adaptability of NN-based approaches but do not provide a benchmark for numerical methods, while we do in our thesis.

### 1.3 Research Goal

Building on the limitations of the HJB PDE method proposed in Forsyth (2022), and the growing potential of neural networks in solving stochastic optimal control problems, Chen et al. (2023) applies the NN-based approach proposed in Li and Forsyth (2019) within the same investment framework as Forsyth (2022), which is based on a 30-year investment horizon and aims to maximize the average annual withdrawal while minimizing risk, where risk is measured by the left tail of the terminal wealth distribution. The NN-based approach involves a dual-network architecture, where one network determines the optimal withdrawal amount, while the other allocates the remaining wealth across different assets. By incorporating continuous time inputs, this approach enables precise and dynamic decision-making. Moreover, the model achieves high accuracy in approximating solutions from the Hamilton-Jacobi-Bellman (HJB) Partial Differential Equation (PDE) framework, with the following advantages:

1. **Data-Driven and Model-Free:** NN framework does not rely on a parametric model. This flexibility allows it to adapt to different training datasets and reduces the risk of model misspecification.
2. **High-Dimensional Applicability:** NN directly learns control strategies, eliminating the need for dynamic programming and mitigating error propagation. Moreover, it scales effectively to high-dimensional asset allocation problems, enhancing both computational efficiency and versatility.

Building on the neural network (NN) framework proposed in Chen et al. (2023), this thesis further investigates the accuracy, robustness, and high-dimensional scalability of NN-based approaches in solving the decumulation and allocation problem. Additionally, we introduce TimeGAN, a generative model for synthesizing realistic financial time-series data.

Our objective function consists of two components: (i) Reward: maximizing total expected withdrawals over the investment horizon; (ii) Risk: minimizing the potential of running out of savings, measured by the left tail in the terminal wealth distribution. A risk aversion parameter  $\kappa$  adjusts the trade-off between these two components. By adjusting the risk aversion parameter, we investigate the computational outcomes of both high-risk (low  $\kappa$  value, making maximizing reward dominate) and low-risk (high  $\kappa$  value, making minimizing risk dominate) investment preferences.

The results reveal that minimizing left-tail risk presents significant computational challenges, particularly for low-risk regions. And we introduce a transfer learning approach that leverages well-trained high-risk models as a foundation for low-risk models to help address this issue.

Beyond assessing the computational accuracy of the neural network framework, it is essential to evaluate its generalization ability across varying market conditions. To investigate the model’s robustness in dynamic investment strategies, we conduct a series of tests using historical data from the Center for Research in Security Prices (CRSP). These tests include out-of-sample and out-of-distribution evaluations, assessing how effectively the model adapts to different data sets. By analyzing performance across diverse financial data sets, we ensure that the NN framework remains reliable and adaptable in real-world applications.

We also explore the high-dimensional scalability of the neural network framework in multi-asset investment scenarios. To assess how the model adapts to diverse assets, we compare the cumulative distribution functions (CDFs) of different bond and stock returns, analyzing how asset characteristics influence withdrawal and allocation strategies. Additionally, we examine heat maps of withdrawal and allocation decisions across various portfolio compositions, providing insights into investment behavior under different market conditions. By applying our NN-based framework to multi-asset portfolios, we evaluate its effectiveness in handling high-dimensional asset allocation problems, ensuring stability and adaptability in complex investment environments.

Furthermore, we introduce TimeGAN as a novel data generation technique to evaluate the adaptability of our NN framework. We conduct an in-depth exploration of TimeGAN’s data generation process, examining how different hyper-parameter settings influence the results of the generated financial time series. To assess our NN framework’s ability to generalize beyond CRSP historical data, we apply our NN framework to these TimeGAN synthetically generated datasets, evaluating its performance on previously unseen data sets.

To summarize, the key contributions of this thesis are as follows:

- Developing a neural network (NN) framework for decumulation and allocation decisions, providing an approximate solution to the constrained stochastic optimal decumulation problem while overcoming computational limitations of traditional PDE-based methods.
- Demonstrating high computational accuracy from minimizing risk perspective by comparing the NN framework training outcomes with the benchmark HJB PDE computational outcomes. We introduce transfer learning to improve training stability in low-risk investment strategies, successfully approximating the PDE benchmark.
- Ensuring robustness by evaluating the NN framework using historical CRSP data and conducting out-of-sample and out-of-distribution tests. Additionally, we leverage TimeGAN-generated synthetic data to assess generalization in unseen data sets.
- Exploring high-dimensional scalability by applying the NN framework to multi-asset portfolios, analyzing asset allocation characteristics and comparing the cumulative distribution functions (CDFs) of bond and stock returns to understand investment patterns across bonds and stocks.
- Utilizing TimeGAN for synthetic data generation to assess the NN model’s adaptability beyond using real CRSP historical datasets. We examine the properties of TimeGAN-generated data and evaluate the NN framework’s performance consistency on these unseen synthetic datasets.

## 1.4 Thesis Outline

Chapter 2 formulates the optimal dynamic allocation problem in defined contribution pension plans, introducing the risk and reward measures and the objective function; Chapter 3 discusses the data generation process, including CRSP historical data, stochastic model calibration, bootstrap resampling, and TimeGAN; Chapter 4 presents the methods for computing optimal dynamic controls, comparing the HJB dynamic programming framework with the neural network formulation; Chapter 5 conducts a computational investigation, focusing on minimizing risk, transfer

learning, heatmap analysis, multi-asset training results, and TimeGAN data generation; finally, Chapter 6 evaluates the robustness of the neural network framework through out-of-sample and out-of-distribution testing, and TimeGAN data testing.

# Chapter 2

## Problem Formulation

### 2.1 Overview

The investment scenario described in [Forsyth \(2022\)](#) focuses on an investor who begins retirement with a specified portfolio wealth. The investment horizon is fixed and divided into equally spaced rebalancing intervals, typically annual. At each rebalancing point, the investor must decide how much to withdraw, e.g., within a predefined range from the portfolio and then allocate the remaining wealth between assets: a broad stock index fund and a constant maturity bond index fund. Between rebalancing periods, the portfolio's wealth changes according to the performance of the underlying assets. At the end of the investment horizon, a final withdrawal is made, and the portfolio is liquidated, resulting in the terminal wealth. The investor is assumed to have other non-fungible assets, such as real estate, which act as a hedge of last resort to cover any accumulated debt ([Pfeiffer et al., 2013](#)). This assumption aligns with the mental bucketing concept proposed by [Shefrin and Thaler \(1988\)](#) and is commonly used in similar problem contexts (see [Forsyth et al. \(2022\)](#)). The investor's goal is to maximize a weighted sum of the expected total withdrawals and the mean of the worst 5% of outcomes in terms of terminal wealth, referred to as the Expected Shortfall (ES) at the 5% level. This chapter outlines the mathematical framework for this optimization problem referred to [Chen et al. \(2023\)](#), which is applicable to both the HJB and NN methods.

### 2.2 Notational Conventions

Let  $\mathcal{T}$  denote the set of discrete, evenly spaced withdrawal and allocation time points

$$\mathcal{T} = \{t_n\}_{n=0}^M, \quad t_n = n\Delta t, \quad \Delta t = \frac{T}{M}. \quad (2.1)$$

At each  $t_i \in \mathcal{T}$ , the investor withdraws  $q_i$  and rebalances the portfolio. At  $t_M = T$ , a final withdrawal  $q_M$  is made, and the portfolio is liquidated.  $S_t$  and  $B_t$  represent the real (i.e. inflation-adjusted) amounts invested in the stock index and a constant maturity bond index, respectively. The total amount in the retirement account at time  $t$ ,  $W_t$  is given by

$$\text{Total wealth} \equiv W_t = S_t + B_t. \quad (2.2)$$

For any function  $f(t)$ , define:

$$f(t_i^+) \equiv \lim_{\epsilon \rightarrow 0^+} f(t_i + \epsilon), \quad f(t_i^-) \equiv \lim_{\epsilon \rightarrow 0^+} f(t_i - \epsilon) \quad (2.3)$$

Let  $X(t) = (S(t), B(t))$  represent the controlled underlying process, with  $x = (s, b)$  denoting the realized state. At each  $t_i$ , the withdrawal  $q_i(\cdot) = q(X(t_i^-), t_i)$  evolves the portfolio wealth:

$$W(t_i^+) = W(t_i^-) - q_i. \quad (2.4)$$

The allocation control  $p_i(\cdot)$  represents the fraction of wealth in stocks after rebalancing:

$$S(t_i^+) = p(X(t_i^+), t_i)W(t_i^+), \quad B(t_i^+) = (1 - p(X(t_i^+), t_i))W(t_i^+). \quad (2.5)$$

Assuming no transaction costs, the control at time  $t_i$  depends only on wealth, simplifying  $p_i(\cdot) = p_i(W_i^+)$ . Thus,  $q_i(\cdot)$  depends on  $W_i^-$ , and  $p_i(\cdot)$  depends on  $W_i^+$ . The control at  $t_i$  is described by  $(q_i(\cdot), p_i(\cdot)) \in \mathcal{Z}(W_i^-, W_i^+, t_i)$ , with constraints:

$$\mathcal{Z}_q(W_i^-, t_i) = \begin{cases} [q_{\min}, q_{\max}], & \text{if } t_i \in \mathcal{T}, W_i^- > q_{\max}, \\ [q_{\min}, W_i^-], & \text{if } t_i \in \mathcal{T}, q_{\min} < W_i^- < q_{\max}, \\ \{q_{\min}\}, & \text{if } t_i \in \mathcal{T}, W_i^- < q_{\min}, \end{cases} \quad (2.6)$$

$$\mathcal{Z}_p(W_i^+, t_i) = \begin{cases} [0, 1], & \text{if } W_i^+ > 0, t_i \in \mathcal{T}, t_i \neq t_M, \\ \{0\}, & \text{if } W_i^+ \leq 0, t_i \in \mathcal{T}, t_i \neq t_M, \\ \{0\}, & \text{if } t_i = t_M, \end{cases} \quad (2.7)$$

$$\mathcal{Z}(W_i^-, W_i^+, t_i) = \mathcal{Z}_q(W_i^-, t_i) \times \mathcal{Z}_p(W_i^+, t_i). \quad (2.8)$$

At each  $t_i$ , we seek the optimal control for all possible combinations of  $(S(t), B(t))$  having the same total wealth (Forsyth, 2022). Hence, the controls for both withdrawal and allocation are formally a function of wealth and time before withdrawal  $(W_i^-, t_i)$ , but for implementation purposes it will be helpful to write the allocation as a function of wealth and time after withdrawal  $(W_i^+, t_i)$ . The admissible control set  $\mathcal{A}$  can be written as

$$\mathcal{A} = \{(q_i, p_i)_{0 \leq i \leq M} : (q_i, p_i) \in \mathcal{Z}(W_i^-, W_i^+, t_i)\}. \quad (2.9)$$

An admissible control  $\mathcal{P} \in \mathcal{A}$  can be written as

$$\mathcal{P} = \{(q_i(\cdot), p_i(\cdot)) : (q_i(\cdot), p_i(\cdot)) \in \mathcal{Z}(W_i^-, W_i^+, t_i), i = 0, \dots, M\}. \quad (2.10)$$

It will sometimes be necessary to refer to the tail of the control sequence at  $[t_n, t_{n+1}, \dots, t_M]$ , which we define as

$$\mathcal{P}_n = \{(q_n(\cdot), p_n(\cdot)), \dots, (p_M(\cdot), q_M(\cdot))\}. \quad (2.11)$$

The essence of the problem, for both the HJB and NN methods outlined in this thesis, will be to find an optimal control  $\mathcal{P}^*$ .

### 2.3 Risk Measure: Expected Shortfall (ES)

Let  $\mathcal{G}(W_T)$  represent the probability density function of the terminal wealth  $W_T$  at time  $t = T$ . For a given confidence level  $0 < \alpha < 1$ , typically set at 5%, let  $W'_\alpha$  satisfy

$$\int_{-\infty}^{W'_\alpha} \mathcal{G}(W_T) dW_T = \alpha. \quad (2.12)$$

i.e.,  $E[\mathbf{1}_{W_T < W'_\alpha}] = \alpha$ .  $W'_\alpha$  can be interpreted as the Value at Risk (VaR) at the level  $\alpha$ . Furthermore, the Expected Shortfall (ES), also known as Conditional Value at Risk (CVaR), is defined as the expected value of  $W_T$  in the worst  $\alpha$  fraction of outcomes<sup>1</sup>. Mathematically, it is given by:

$$ES_\alpha = \frac{\int_{-\infty}^{W'_\alpha} W_T \mathcal{G}(W_T) dW_T}{\alpha}. \quad (2.13)$$

This metric provides a more comprehensive risk assessment by considering the magnitude of potential losses beyond the VaR threshold. A higher Expected Shortfall (ES) is preferable, as equation

<sup>1</sup>Usually, CVAR is defined in terms of losses, not final wealth. Hence, usually CVAR is the negative of ES.

(2.13) reflects greater final wealth rather than larger losses. For computational convenience, ES in Rockafellar and Uryasev (2000) is defined as:

$$ES_\alpha = \sup_{W'} \mathbb{E} \left[ W' + \frac{1}{\alpha} \min(W_T - W', 0) \right]. \quad (2.14)$$

Under a given control policy  $\mathcal{P}$  and an initial state  $X_0$ , this definition extends to:

$$ES_\alpha(X_0^-, t_0^-) = \sup_{W'} \mathbb{E}_{\mathcal{P}}^{X_0^-, t_0^-} \left[ W' + \frac{1}{\alpha} \min(W_T - W', 0) \right]. \quad (2.15)$$

The candidate values of  $W'$  are selected from the set of possible terminal wealth values  $W_T$ . Here,  $ES_\alpha(X_0^-, t_0^-)$  represents the expected shortfall as observed at  $t_0^-$ , with  $W'$  remaining fixed throughout the investment horizon. Importantly, this approach ensures time consistency in the ES calculation, distinguishing it from time-inconsistent versions of expected shortfall policies (Forsyth, 2020; Strub et al., 2019).

## 2.4 Reward Measure: Total Expected Withdrawals (EW)

We quantify the reward using the expected total amount of withdrawals. Formally, the expected withdrawals (EW) are defined as:

$$EW(X_0^-, t_0^-) = E_{\mathcal{P}}^{X_0^-, t_0^-} \left[ \sum_{n=0}^M q_n \right]. \quad (2.16)$$

This represents the expected sum of withdrawals over the entire investment horizon under a given control policy.

## 2.5 Objective Function

In this section, we introduce the common objective function shared by both the HJB and NN methods. Since an increase in Expected Withdrawals (EW) generally leads to a reduction in Expected Shortfall (ES), we identify Pareto optimal solutions for this multi-objective problem. Given a scalarization parameter  $\kappa$ , we aim to determine the optimal control  $\mathcal{P}_0$  that maximizes the following expression:

$$EW(X_0^-, t_0^-) + \kappa ES_\alpha(X_0^-, t_0^-). \quad (2.17)$$

We define Equation (2.17) as the pre-commitment EW-ES problem, denoted as  $(PCEE_{t_0}(\kappa))$ , and formally express it as follows:

$(PCEE_{t_0}(\kappa)) :$

$$J(s, b, t_0^-) = \sup_{W'} \sup_{\mathcal{P}_0 \in \mathcal{A}} \left\{ \mathbb{E}_{\mathcal{P}_0}^{X_0^-, t_0^-} \left[ \sum_{i=0}^M q_i + \kappa \left( W' + \frac{1}{\alpha} \min(W_T - W', 0) \right) + \underbrace{\epsilon W_T}_{\text{stabilization}} \right] \right\} \left| X(t_0^-) = (s, b) \right\}$$

$$\text{subject to } \begin{cases} (S_t, B_t) \text{ are stock and bond account amount (inflation adjusted), which will be} \\ \text{detailed in Section 3.3; } t \notin \mathcal{T}, \\ W_i^+ = S_i^- + B_i^- - q_i, \quad X_i^+ = (S_i^+, B_i^+), \\ S_i^+ = p_i(\cdot)W_i^+, \quad B_i^+ = (1 - p_i(\cdot))W_i^+, \\ (q_i(\cdot), p_i(\cdot)) \in \mathcal{Z}(W_i^-, W_i^+, t_i), \\ i = 0, \dots, M, \quad t_i \in \mathcal{T}. \end{cases} \quad (2.18)$$

The stabilization term  $\epsilon W_T$  is introduced to address potential ill-posedness in cases where  $W_t \gg W'$  as  $t \rightarrow T$ . This term has negligible influence on the optimal values of EW, ES, or other summary statistics when  $|\epsilon| \ll 1$ . The objective function in Equation (2.18) serves as the foundation for the value function in the HJB framework and as the loss function in the NN-based approach.

**Remark 1.** *Although Equation (2.18) is formally the pre-commitment policy, which is time inconsistent, we assume that the investor follows the optimal policy for  $t > 0$  with fixed value of  $W^*$ ,*

$$W^*(s, b) = \arg \max_{W^*} \left\{ \sup_{\mathbb{P}_0 \in \mathcal{A}} \mathbb{E}_{\mathbb{P}_0}^{X_0^-, t_0^-} \left[ \sum_{i=0}^M q_i + \kappa \left( W^* + \frac{1}{\alpha} \min(W_T - W^*, 0) \right) \mid X(t_0^-) = (s, b) \right] \right\}. \quad (2.19)$$

*This policy is trivially time consistent, which is implementable. See Forsyth (2020); Strub et al. (2019) for a discussion of induced time consistent policies.*

# Chapter 3

## Data Generation

### 3.1 Overview

For the computational analysis in this study, we utilize monthly return data from the Center for Research in Security Prices (CRSP). Specifically, we employ the CRSP 10-year U.S. Treasury index to represent the bond asset and the CRSP cap-weighted total return index for the equity asset. Given that retirees are primarily focused on maintaining real spending power rather than nominal values, we adjust these indices for inflation using the US Consumer Price Index (CPI) from CRSP. This market data is applied in three distinct ways in our subsequent analyses: Stochastic model calibration, Bootstrap resampling, and TimeGAN-generated data. For Stochastic model calibration and Bootstrap resampling, we use the same methods as in [Ni et al. \(2022\)](#).

### 3.2 CRSP Historical Data

The dataset from [Center for Research in Security Prices \(CRSP\) \(2024\)](#) provides a comprehensive collection of financial market data widely used in empirical research and portfolio analysis. It includes a range of equity and fixed-income assets, allowing for the study of market trends, risk-return characteristics, and asset allocation strategies over extended historical periods.

In this thesis, we focus on five key CRSP-tracked assets, which represent a diverse mix of equity and fixed-income instruments. These assets are commonly used in portfolio construction and financial modeling.

#### 1. CRSP 10-Year T-Bond Returns

- Represents long-term U.S. government securities, offering a stable, lower-risk investment option. <sup>1</sup>
- Used as a benchmark for risk-free long-term rates and often serves as a safe-haven asset during periods of economic unseent.
- Influenced by interest rate changes, inflation expectations, and monetary policy.

#### 2. CRSP 30-Day & 90-Day T-Bill Returns

- Represents short-term U.S. Treasury bills, with 30-day and 90-day maturities.
- Commonly used as proxies for risk-free rates, influencing asset pricing models and discounting future cash flows.
- Typically has lower volatility compared to equities, making it a core component of low-risk portfolios.

#### 3. CRSP Value-Weighted Index

---

<sup>1</sup>The CRSP 10-Year T-Bond Return is constructed in the following way. At the beginning of each month, a 10-year bond is purchased. At the end of the month, the bond is sold. Return is computed with total capital gain and interest payments. Then, a new 10-year bond is purchased at the beginning of the next month.

- A market-cap-weighted total return index <sup>2</sup> that tracks the performance of U.S. equities, where larger companies have a greater influence on the index return.
- Reflects broad market trends, commonly used as a benchmark for stock market performance.
- More sensitive to large-cap stocks, as weighting is proportional to company size.

#### 4. CRSP Equal-Weighted Index

- Unlike the value-weighted index, the equal-weighted total return index assigns the same weight to each stock, regardless of market capitalization.
- Reduces the dominance of large firms, providing a more balanced representation of stock market performance.
- Tends to outperform value-weighted indexes over long horizons due to a higher weighting on small-cap stocks, which historically offer higher returns but also greater volatility.

### 3.3 Stochastic Model Calibration

Both stock index and bond index are modeled with correlated jump diffusion models, in line with [MacMinn et al. \(2014\)](#). These parametric stochastic differential equations (SDEs) allow us to model non-normal asset returns. The SDEs are used in solving the HJB PDE, and generating training data with Monte Carlo (MC) simulations in the proposed NN framework.

The foundation of the jump-diffusion model is a standard geometric Brownian motion, characterized by a drift rate of  $\mu^s$  and a volatility of  $\sigma^s$ . To account for jump effects, an additional term is incorporated, along with a compensator to maintain the expected drift rate. In the context of stocks, this gives the following stochastic differential equation (SDE) that describes how the amount in the stock account  $S_t$  (inflation adjusted) evolves between rebalancing times:

$$\frac{dS_t}{S_{t-}} = (\mu^s - \lambda_\xi^s \gamma_\xi^s) dt + \sigma^s dZ^s + d \left( \sum_{i=1}^{\pi_t^s} (\xi_i^s - 1) \right), \quad t \in (t_i, t_{i+1}) \quad (3.1)$$

where  $dZ^s$  represents the increment of a Wiener process, while  $\pi_t^s$  follows a Poisson process with a positive intensity parameter  $\lambda_\xi^s$ . When a jump occurs, the stock value updates as  $S_t = \xi^s S_{t-}$ , where  $\xi^s$  represents the jump multiplier, and  $S_{t-}$  is the stock value immediately before the jump at time  $t$ . The logarithm of the jump multiplier,  $\log(\xi^s)$ , is modeled by a double exponential distribution, as described in references [Kou \(2002\)](#); [Kou and Wang \(2004\)](#). The jump can be either upward or downward, occurring with probabilities  $u^s$  and  $1 - u^s$ , respectively. Defining  $y = \log(\xi^s)$ , its probability density function is given by:

$$f^s(y) = u^s \eta_1^s e^{-\eta_1^s y} \mathbf{1}_{y \geq 0} + (1 - u^s) \eta_2^s e^{\eta_2^s y} \mathbf{1}_{y < 0}. \quad (3.2)$$

Additionally, we define the expected jump adjustment term as:

$$\gamma_\xi^s = \mathbb{E}[\xi^s - 1] = \frac{u^s \eta_1^s}{\eta_1^s - 1} + \frac{(1 - u^s) \eta_2^s}{\eta_2^s + 1} - 1. \quad (3.3)$$

Each  $\xi_i^s$  is independently and identically distributed (i.i.d.), strictly positive, and follows a given distribution (3.2). Additionally, it is assumed that  $\xi_i^s$ ,  $\pi_i^s$ , and  $Z^s$  are mutually independent.

Following the approach in [MacMinn et al. \(2014\)](#); [Lin et al. \(2015\)](#); [Forsyth et al. \(2024\)](#), we represent the constant maturity real bond index through a jump diffusion process. Let  $B_{t-} = B(t - \epsilon)$ , where  $\epsilon \rightarrow 0^+$ , denote the bond index value just before time  $t$ . Between rebalancing periods, the evolution of the bond account  $B_t$  is governed by:

$$\frac{dB_t}{B_{t-}} = \left( \mu^b - \lambda_\xi^b \gamma_\xi^b + \mu_c^b \mathbf{1}_{B_{t-} < 0} \right) dt + \sigma^b dZ^b + d \left( \sum_{i=1}^{\pi_t^b} (\xi_i^b - 1) \right), \quad t \in (t_i, t_{i+1}). \quad (3.4)$$

---

<sup>2</sup>A total return index reflects the performance of an investment by accounting for both capital gains and the reinvestment of income such as dividends or interest payments.

The terms in Equation (3.4) are defined similarly to those in Equation (3.1). The Poisson process  $\pi_t^b$  has a non-negative intensity parameter  $\lambda_\xi^b$ , and the expected jump size is given by  $\gamma_\xi^b = \mathbb{E}[\xi^b - 1]$ . The logarithm of the jump multiplier,  $y = \log(\xi^b)$ , follows the same double exponential distribution as in Equation (3.2), denoted by  $f^b(y)$ , but with parameters  $u^b$ ,  $\eta_1^b$ , and  $\eta_2^b$  specific to bonds. It is assumed that  $\xi_t^b$ ,  $\pi_t^b$ , and  $Z^b$  are mutually independent, consistent with the assumptions for the stochastic differential equation of  $S_t$ . The term  $\mu_c^b \mathbf{1}_{B_t < 0}$  represents the borrowing spread, which is assumed to be non-negative.

The diffusion processes of the two assets are correlated with a coefficient  $\rho_{sb}$ , meaning that  $dZ^s \cdot dZ^b = \rho_{sb} dt$ . However, the jump processes are assumed to be independent. For a detailed justification of this market model, refer to Forsyth (2022).

The term *simulated data* in this thesis refers to data generated using parametric stochastic models (SDEs), with parameters calibrated to CPI adjusted CRSP data spanning from 1926:1 to 2019:12. To calibrate the model parameters, we employ a two-step procedure. First, we estimate the diffusion and jump parameters using historical asset return data. The diffusion components ( $\mu^s, \sigma^s$  for stocks and  $\mu^b, \sigma^b$  for bonds) are obtained via maximum likelihood estimation (MLE) on log returns, assuming a geometric Brownian motion framework. The jump intensities ( $\lambda^s, \lambda^b$ ) and distributions ( $\eta_1, \eta_2, u$ ) are inferred using a threshold-based approach (Cont and Mancini, 2011; Dang and Forsyth, 2016; Mancini, 2009) to detect discrete jumps. Once the jump times are identified, we fit a double-exponential distribution to the observed jump sizes, estimating  $\eta_1, \eta_2$ , and  $u$  separately for stocks and bonds. The correlation between asset returns,  $\rho_{sb}$ , is computed from the Wiener process increments after removing identified jumps. The final calibrated parameters are presented in Table 3.1. For further details on the jump detection method, see Dang and Forsyth (2016).

CRSP	$\mu^s$	$\sigma^s$	$\lambda^s$	$u^s$	$\eta_1^s$	$\eta_2^s$	$\rho_{sb}$
	0.0877	0.1459	0.3191	0.2333	4.3608	5.504	0.04554
10-year Treasury	$\mu^b$	$\sigma^b$	$\lambda^b$	$u^b$	$\eta_1^b$	$\eta_2^b$	$\rho_{sb}$
	0.0239	0.0538	0.3830	0.6111	16.19	17.27	0.04554

Table 3.1: *Calibrated (annualized) parameters for double exponential jump diffusion model. CPI adjusted CRSP US Total Market Index and CRSP US 10-year treasury, also inflation adjusted. Data from 1926:1 to 2019:12.*

### 3.4 Bootstrap Resampling

The term *resampled data* used in this thesis is generated via the stationary block bootstrap method, as described by Dichtl et al. (2016); Patton et al. (2009); Politis and Romano (1994); Politis and White (2004). This approach involves drawing randomly sampled blocks of varying lengths, with replacement, from the original CRSP dataset. The block sizes follow a geometric distribution with a predefined expected value. To maintain the correlation structure between asset returns, blocks from both stock and bond series are sampled simultaneously. This technique effectively reshuffles the original data while preserving short-term dependencies within each block, allowing for repeated resampling to generate multiple possible return paths. The method helps account for potential serial correlation in financial time series. More details, including pseudo-code for implementation, can be found in Forsyth and Vetzal (2019).

A crucial parameter in block resampling is the expected block size. Following the algorithm in Patton et al. (2009), the optimal block size for stock and bond returns is determined separately (see Table 3.2). In this dataset, a reasonable expected block size for paired resampling is approximately three months, as suggested by Forsyth (2022). Additionally, we conduct sensitivity tests by varying block sizes between 1 and 12 months in numerical experiments.

Data	Optimal expected block size $\hat{b}$ (months)
CRSP US 10-year treasury	4.2
CPI adjusted CRSP US Total Market Index	3.1

Table 3.2: Optimal expected blocksize  $\hat{b} = 1/\nu$ , from [Patton et al. \(2009\)](#). The blocksize is a draw from a geometric distribution with  $Pr(b = k) = (1 - \nu)^{k-1}\nu$ .

## 3.5 TimeGAN

Any data set referred to in this thesis as *TimeGAN data* is generated by using the TimeGAN (Time-series Generative Adversarial Networks) framework. TimeGAN is a state-of-the-art framework designed to generate realistic time-series data by combining the strengths of adversarial training and embedding learning. Unlike traditional GANs, which are primarily used for generating static data such as images, TimeGAN is specifically tailored to handle the temporal dependencies and complex dynamics inherent in time-series data, making it particularly suitable for financial applications ([Yoon et al., 2019](#)).

In this section, we present a detailed mathematical explanation of how TimeGAN is used to generate synthetic CRSP data, as referenced in [Yoon et al. \(2019\)](#). Below, we outline the key components and training objectives of TimeGAN, incorporating mathematical formulations to elucidate its architecture and training process.

### 3.5.1 TimeGAN Framework

Let  $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_L\}$  represent a real time-series dataset, where  $L$  is the sequence length and each  $\mathbf{x}_l \in \mathbb{R}^n$  is an  $n$ -dimensional observation at sequence index  $l$ . In the context of CRSP data,  $\mathbf{x}_l$  represents the 2-dimensional inflation-adjusted stock and bond returns at index  $l$ . The goal of TimeGAN is to learn a generator  $\mathcal{G}$  that can produce synthetic time-series data  $\hat{\mathcal{X}} = \{\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \dots, \hat{\mathbf{x}}_L\}$  that preserves the temporal dynamics properties of  $\mathcal{X}$ .

#### Model Components

In TimeGAN, temporal features refer to attributes that vary over time. These features capture the dynamic nature of the data and influence how patterns evolve across different time steps. For instance, in CRSP data, the monthly real returns of stocks and bonds are temporal features, as they fluctuate based on market conditions. The goal of TimeGAN is to learn the temporal dependencies within real-world time-series data, effectively capturing the underlying dynamic relationships. By doing so, the model can generate synthetic data that preserves the temporal structures of the original dataset.

TimeGAN consists of four Recurrent Neural Networks, each with a specific role in the training process (Refer to [Table 3.3](#) for description of each symbol during the framework training process):

- **Encoder Network ( $e$ ):** The encoder maps the real time-series  $\mathbf{x}_{1:L}$  into latent space representations  $\mathbf{h}_{1:L}$  to capture the hidden features of the data:

$$\mathbf{h}_{1:L} = e(\mathbf{x}_{1:L}; \theta_e) \quad (3.5)$$

- **Recovery Network ( $r$ ):** The recovery network reconstructs the original data from latent representations  $\mathbf{h}_{1:L}$ :

$$\tilde{\mathbf{x}}_{1:L} = r(\mathbf{h}_{1:L}; \theta_r) \quad (3.6)$$

- **Generator Network ( $g$ ):** The generator produces latent representations  $\hat{\mathbf{h}}_{1:L}$  from random vectors  $\mathbf{z}_{1:L}$  and generates synthetic latent representations:

$$\hat{\mathbf{h}}_l = g(\hat{\mathbf{h}}_{l-1}, \mathbf{z}_l; \theta_g) \quad (3.7)$$

- **Discriminator Network ( $d$ ):** The discriminator’s output is a probability indicating whether the input is real or synthetic. The discriminator distinguishes between real latent representations  $\mathbf{h}_l$  and synthetic ones  $\hat{\mathbf{h}}_l$ :

$$y_l = d(\mathbf{h}_l; \theta_d) \in [0, 1], \quad \hat{y}_l = d(\hat{\mathbf{h}}_l; \theta_d) \in [0, 1] \quad (3.8)$$

Symbol	Description
$\mathbf{x}_{1:L}$	Real data. In our case, it is the 2-dimensional vector of inflation-adjusted stock and bond returns.
$\mathbf{h}_{1:L}$	Latent representations of real data from the Encoder Network.
$\tilde{\mathbf{x}}_{1:L}$	Reconstructed real data from the Recovery Network.
$\hat{\mathbf{h}}_{1:L}$	Latent representations of synthetic data from the Generator Network.
$\mathbf{z}_{1:L}$	Random vectors used as input for the Generator Network. In our case, it is the synthetic source data sampling from the $[0, 1]$ uniform distribution.
$y_l$	Output of the Discriminator Network for real data.
$\hat{y}_l$	Output of the Discriminator Network for synthetic data.
$\theta_e$	Trainable parameters in the Encoder Network, which maps the real time-series data into a latent space representation.
$\theta_r$	Trainable parameters in the Recovery Network, which reconstructs the real time-series data from its latent space representation.
$\theta_g$	Trainable parameters in the Generator Network, which learns to produce synthetic latent representations from random vectors.
$\theta_d$	Trainable parameters in the Discriminator Network, which is optimized to distinguish between real and synthetic latent representations.
$p$	The distribution of the real dataset.
$\hat{p}$	The distribution of the synthetic dataset generated by the Generator Network.

Table 3.3: *Notation and Description of Variables in TimeGAN: An overview of the key symbols used in the TimeGAN framework, detailing their roles in capturing temporal dependencies and generating synthetic time-series data. It includes representations of real and synthetic features, latent embeddings, trainable parameters for each network component, and the discriminator’s decision variables.*

### Loss Function

TimeGAN employs a multi-task learning framework to encode, generate, and iterate over time-series data by combining reconstruction loss, adversarial loss, and embedding loss to jointly optimize the model components.

- **Reconstruction Loss ( $\mathcal{L}_R$ ):** The reconstruction loss measures the discrepancy between the reconstructed real sequences  $\tilde{\mathbf{x}}_{1:L}$  from the latent representations  $\mathbf{h}_{1:L}$  and the original real sequences  $\mathbf{x}_{1:L}$ . Minimizing this loss ensures that the Recovery Network ( $r$ ) effectively reconstructs the original time-series data from its latent representations.

$$\mathcal{L}_R = \mathbb{E}_{\mathbf{x}_{1:L} \sim p} \left[ \sum_l \|\mathbf{x}_l - \tilde{\mathbf{x}}_l\|_2 \right] \quad (3.9)$$

- **Adversarial Loss ( $\mathcal{L}_A$ ):** The adversarial loss measures how well the Discriminator Network ( $d$ ) distinguishes between real and synthetic latent representations. By maximizing the loss,

it encourages the Discriminator Network ( $d$ ) to assign high probabilities to real data  $y_l$  and low probabilities to synthetic data  $\hat{y}_l$ , while the Generator Network ( $g$ ) aims to minimize this loss by producing more realistic latent representations.

$$\mathcal{L}_A = \mathbb{E}_{\mathbf{x}_{1:L} \sim p} \left[ \sum_l \log y_l \right] + \mathbb{E}_{\mathbf{x}_{1:L} \sim \hat{p}} \left[ \sum_l \log(1 - \hat{y}_l) \right] \quad (3.10)$$

- **Embedding Loss ( $\mathcal{L}_E$ ):** The embedding loss explicitly measures the discrepancy between the conditional distributions:  $\hat{p}(\mathbf{H}_l | \mathbf{H}_{1:l-1})$  and  $p(\mathbf{H}_l | \mathbf{H}_{1:l-1})$ . By minimizing this loss, it encourages the Generator Network ( $g$ ) to accurately predict the latent representation of the next time step for synthetic data.

$$\mathcal{L}_E = \mathbb{E}_{\mathbf{x}_{1:L} \sim p} \left[ \sum_l \|\mathbf{h}_l - g(\mathbf{h}_{l-1}, \mathbf{z}_l)\|_2 \right] \quad (3.11)$$

## Optimization

The optimization process in TimeGAN involves training the Encoder ( $e$ ), Recovery ( $r$ ), Generator ( $g$ ), and Discriminator ( $d$ ) Networks to jointly encode, generate, and iterate over time-series data. The optimization is divided into two key steps:

- **Training the Encoder and Recovery Networks:** The Encoder ( $e$ ) and Recovery ( $r$ ) Networks are trained using a combination of reconstruction loss ( $\mathcal{L}_R$ ) and embedding loss ( $\mathcal{L}_E$ ). The optimization goal is to ensure that the latent representations accurately reconstruct the original time-series data while preserving temporal relationships. This optimization is given by:

$$\min_{\theta_e, \theta_r} (\lambda \mathcal{L}_E + \mathcal{L}_R) \quad (3.12)$$

Here,  $\lambda \geq 0$  is a hyper-parameter that balances the embedding loss and the reconstruction loss. Here we choose  $\lambda = 0.01$  for training.

- **Adversarial Training of Generator and Discriminator Networks:** The Generator ( $g$ ) and Discriminator ( $d$ ) Networks are trained in an adversarial framework, where the Generator Network ( $g$ ) tries to minimize both the embedding loss ( $\mathcal{L}_E$ ) and the adversarial loss ( $\mathcal{L}_A$ ), while the Discriminator Network ( $d$ ) maximizes the adversarial loss to distinguish real from synthetic data. The optimization can be expressed as:

$$\min_{\theta_g} (\eta \mathcal{L}_E + \max_{\theta_d} \mathcal{L}_A) \quad (3.13)$$

Here,  $\eta \geq 0$  is another hyper-parameter that balances the embedding loss and the adversarial loss. Here we choose  $\eta = 100$  for training.

For the selection of  $\lambda$  and  $\eta$ , a detailed investigation is presented in Section 5.6.2. Figure 3.1 illustrates the overall TimeGAN framework training process, encompassing both the block diagram and the training scheme.

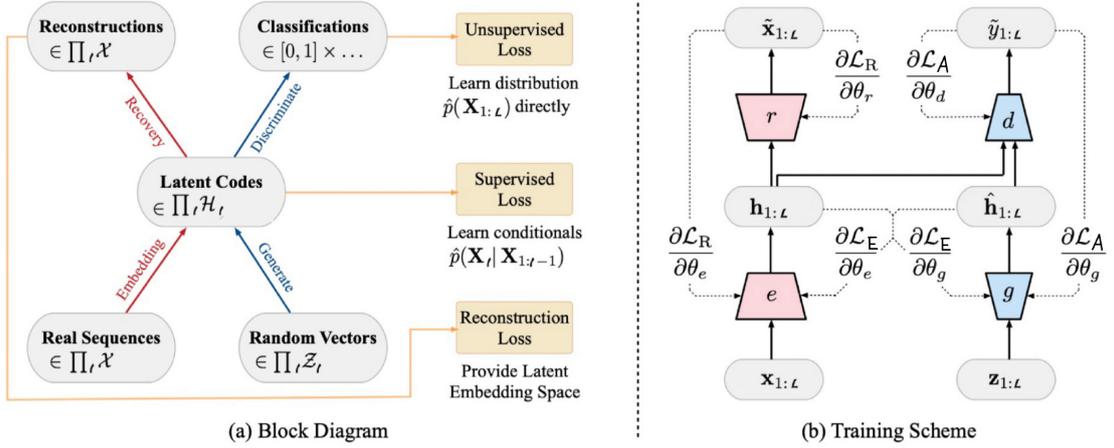


Figure 3.1: *TimeGAN Framework Training Process.* (a) Block diagram of component functions and objectives. (b) Training scheme; solid lines indicate forward propagation of data, and dashed lines indicate backpropagation of gradients. (Yoon et al., 2019)

### TimeGAN Synthetic Data Generation Process

After training, the model effectively captures the temporal dependencies of CRSP real returns. We then utilize the Generator Network ( $g$ ) and the Recovery Network ( $r$ ) to generate TimeGAN synthetic data, where the synthetic source data is drawn from a uniform distribution over  $[0, 1]$ . Data flow (3.14) is the synthetic data generation process.

$$\mathbf{z}_{1:L} \xrightarrow{g} \hat{\mathbf{h}}_{1:L} \xrightarrow{r} \tilde{\mathbf{x}}_{1:L} \quad (\text{Generated data flow}) \quad (3.14)$$

- $\mathbf{z}_{1:L}$ : Random vectors sampled from a uniform distribution, serving as the input to the Generator Network.
- $g$ : The Generator Network, which maps the random vectors to synthetic latent representations.
- $\hat{\mathbf{h}}_{1:L}$ : Synthetic latent representations generated by the Generator Network.
- $r$ : The Recovery Network, which reconstructs the final synthetic time-series data from the latent representations.
- $\tilde{\mathbf{x}}_{1:L}$ : The generated synthetic time-series data produced by the Recovery Network.

### 3.5.2 CRSP TimeGAN Data Generation for Optimal Control Allocation Problem

The original CPI adjusted CRSP data comprises 1176 months of joint stock and bond return data<sup>3</sup> from January 1926 to December 2023 (CRSP 10-Year T-Bond Returns & CRSP Value-Weighted Index). To meet the input requirements of TimeGAN, we segmented the 1176 months of data into multiple sequences of length  $L$ , with each sequence containing  $L$  months of stock and bond return data. The sequence length corresponds to the block size used in the bootstrap resampling method described in Section 3.4. The impact of different sequence lengths on data generation performance will be evaluated in the experiments presented in Section 5.6.2. The specific steps are as follows:

#### 1. Data Segmentation:

The original data can be represented as:

$$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{1176}], \quad \mathbf{x}_k = (r_k^s, r_k^b), \quad (3.15)$$

<sup>3</sup>The asset data used in our TimeGAN framework consists of real returns (i.e., inflation-adjusted returns), rather than the original asset price levels utilized in the original code from Yoon et al. (2019).

where  $r_k^s$  and  $r_k^b$  represent the stock and bond returns at month  $k$ , respectively. We segment the 1176 months of data into multiple sequences of length  $L$ , with each sequence represented as:

$$\mathbf{X}_k = [\mathbf{x}_k, \mathbf{x}_{k+1}, \dots, \mathbf{x}_{k+L-1}], \quad k = 1, 2, \dots, 1176 - (L - 1). \quad (3.16)$$

## 2. TimeGAN Parameters Training and Data Generating:

These sequences of length  $L$  are input into the TimeGAN framework as real data for training. TimeGAN learns the temporal dependencies of these sequences to obtain an appropriate set of network parameters  $\theta_e, \theta_r, \theta_g, \theta_d$ . After training the model, we apply the Generator Network and the Recovery Network to random vectors to generate the synthetic data sequences of length  $L$ .

## 3. Data Recomposition:

After generating synthetic data, we concatenate multiple sequences of length  $L$  end-to-end to form a monthly return series. Assuming we generate  $M$  sequences of length  $L$ , the final synthetic data can be represented as:

$$\hat{\mathbf{X}} = [\hat{\mathbf{X}}_1, \hat{\mathbf{X}}_2, \dots, \hat{\mathbf{X}}_M], \quad (3.17)$$

where each  $\hat{\mathbf{X}}_k$  is a  $L \times 2$  matrix representing  $L$  months of generated stock and bond returns:

$$\hat{\mathbf{X}}_k = [\hat{\mathbf{x}}_k, \hat{\mathbf{x}}_{k+1}, \dots, \hat{\mathbf{x}}_{k+L-1}], \quad \hat{\mathbf{x}}_k = (\hat{r}_k^s, \hat{r}_k^b), \quad (3.18)$$

By concatenating  $M$  synthetic sequences in chronological order, we generate a time series of length  $M \cdot L$ . The final generated time series can be represented as:

$$\hat{\mathbf{X}} = [\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \dots, \hat{\mathbf{x}}_{M \cdot L}], \quad \hat{\mathbf{x}}_k = (\hat{r}_k^s, \hat{r}_k^b), \quad (3.19)$$

where each  $\hat{\mathbf{x}}_k$  represents the synthetic stock and bond returns at month  $k$ .

## 4. Generating Training Trajectory Paths in Optimal Control Allocation Problem:

In our Optimal Control Allocation Problem (2.18), since each path requires 360 monthly returns  $\hat{\mathbf{x}}_k$  (30 years), we set  $M \cdot L = 360$  in Equation (3.19) to generate  $360/L$  synthetic sequences of length  $L$ . Suppose we require  $N$  training paths, we need a total of  $360N/L$  TimeGAN synthetic sequences.

Table 3.4 presents the hyper-parameter settings of the TimeGAN model for the Optimal Control Allocation Problem. A detailed investigation of the TimeGAN synthetic data generation performance is conducted in Section 5.6.

Parameter Name	Parameter Value	Description
Sequence Length	6	The length of each time series, representing 6 months of stock and bond return data. The selected sequence length is referenced from <a href="#">van Staden et al. (2024)</a> .
Network Module	GRU	The neural network module used, GRU (Gated Recurrent Unit) for capturing temporal dependencies.
Hidden Layer Dimension	24	The dimension of the hidden layer, determining the model's expressive power.
Number of Layers	5	The number of layers in the neural network, affecting the model's complexity and fitting capability.
Training Iterations	20000	The total number of training iterations, ensuring the model fully learns the data distribution.
Batch Size	128	The number of samples used in each training batch, affecting training speed and model stability.

Table 3.4: *TimeGAN Framework Training Hyper-parameter Settings for CPI Adjusted CRSP Data (CRSP 10-Year T-Bond Returns & CRSP Value-Weighted Index, from 1926:1 to 2023:12) in Optimal Control Allocation Problem.*

# Chapter 4

## Methods for Computing Optimal Dynamic Controls

### 4.1 Overview

In this chapter, we explore computational methods for Optimal Dynamic Controls, focusing on two methods: the HJB Dynamic Programming Optimization Framework, proposed in Forsyth (2022), and the Neural Network Formulation, introduced in Chen et al. (2023). We will employ these two frameworks to solve the pre-commitment EW-ES problem ( $PCEE_{t_0}(\kappa)$ ) (2.18).

### 4.2 HJB Dynamic Programming Optimization Framework

The HJB framework applies dynamic programming by breaking the problem into sub-problems at each time step and solving them in a backward manner. For clarity, we provide a summary of the algorithm from Forsyth (2022) here.

#### 4.2.1 Deriving Auxiliary Function for $PCEE_{t_0}(\kappa)$

The HJB framework starts by defining auxiliary functions derived from the objective function (2.18) and the underlying stochastic processes. An equivalent problem is then formulated and solved to determine the optimal value function.

To establish the HJB solution, we begin by interchanging the  $\sup_{P_0}$  and  $\sup_{W^*}$  operators:

$$J(s, b, t_0^-) = \sup_{W^*} \sup_{P_0 \in \mathcal{A}} \mathbb{E}_{P_0}^{X_0^-, t_0^-} \left[ \sum_{n=0}^M q_n + \kappa \left( W^* + \frac{1}{\alpha} \min(W_T - W^*, 0) \right) + \epsilon W_T \mid X(t_0^-) = (s, b) \right]. \quad (4.1)$$

The auxiliary function, which must be computed within the dynamic programming framework at each time step  $t_n$ , provides an associated strategy for any  $t_n > 0$ , aligning with the solution of ( $PCEE_{t_0}(\kappa)$ ) (2.18) for a given  $W^*$ . For a comprehensive discussion on pre-commitment and time-consistent ES strategies, we refer to Forsyth (2020), which also presents a proof outlining the derivation steps of the auxiliary function from (4.1). Incorporating  $W^*$  into the state space results in the expanded state variable  $\hat{X} = (s, b, W^*)$ .

Define the problem domain  $\Omega = [0, \infty) \times (-\infty, +\infty) \times (-\infty, +\infty) \times [0, \infty)$ . The auxiliary function  $V(s, b, W^*, t) \in \Omega$  is then defined as:

$$V(s, b, W', t_n^-) = \sup_{\mathcal{P}_n \in \mathcal{A}_n} \left\{ \mathbb{E}_{\mathcal{P}_n}^{\hat{X}_n^-, t_n^-} \left[ \sum_{i=n}^M q_i + \kappa \left( W' + \frac{1}{\alpha} \min(W_T - W', 0) \right) + \underbrace{\epsilon W_T}_{\text{stabilization}} \right] \mid \hat{X}(t_n^-) = (s, b, W') \right\}$$

$$\text{subject to } \begin{cases} (S_t, B_t) \text{ follow processes (3.1) and (3.4); } & t \notin \mathcal{T}, \\ W_i^+ = S_i^- + B_i^- - q_i, & \hat{X}_i^+ = (S_i^+, B_i^+), \\ S_i^+ = p_i(\cdot)W_i^+, & B_i^+ = (1 - p_i(\cdot))W_i^+, \\ (q_i(\cdot), p_i(\cdot)) \in \mathcal{Z}(W_i^-, W_i^+, t_i), \\ i = 0, \dots, M, & t_i \in \mathcal{T}. \end{cases} \quad (4.2)$$

## 4.2.2 Applying Dynamic Programming at Rebalancing Times

The dynamic programming principle is applied at each  $t_n \in \mathcal{T}$  based on Equation (4.2). As is customary, the optimal control must be determined in reverse time order. To achieve this, we decompose the  $\sup_{\mathcal{P}_n}$  operator into  $\sup_{q \in \mathcal{Z}_q} \sup_{p \in \mathcal{Z}_p(w^-, q, t)}$ .

$$\begin{aligned} V(s, b, W^*, t_n^-) &= \sup_{q \in \mathcal{Z}_q} \sup_{p \in \mathcal{Z}_p(w^-, q, t)} \left\{ q + V((w^- - q)p, (w^- - q)(1 - p), W^*, t_n^+) \right\} \\ &= \sup_{q \in \mathcal{Z}_q} \left\{ q + \left[ \sup_{p \in \mathcal{Z}_p(w^-, q, t)} V((w^- - q)p, (w^- - q)(1 - p), W^*, t_n^+) \right] \right\} \end{aligned} \quad (4.3)$$

Let  $\bar{V}$  denote the upper semi-continuous envelope of  $V$ , which will have already been computed as the algorithm progresses backward through time. The optimal allocation control  $p_n(w, W^*)$  at time  $t_n$  is determined from

$$p_n(w, W^*) = \begin{cases} \arg \max_{p' \in [0, 1]} \bar{V}(wp', w(1 - p'), W^*, t_n^+), & w > 0; t_n \neq t_M \\ 0, & w \leq 0 \text{ or } t_n = t_M \end{cases} \quad (4.4)$$

The control  $q$  is then determined from

$$q_n(w, W^*) = \arg \max_{q' \in \mathcal{Z}_q} \left\{ q' + \bar{V}((w - q')p_n(w - q', W^*), (w - q')(1 - p_n(w - q', W^*)), W^*, t_n^+) \right\} \quad (4.5)$$

Using these controls for  $t_n$ , the solution is then advanced backwards across time from  $t_n^+$  to  $t_n^-$  by

$$V(s, b, W^*, t_n^-) = q_n(w^-, W^*) + \bar{V}(w^+ p_n(w^+, W^*), w^+(1 - p_n(w^+, W^*)), W^*, t_n^+) \quad (4.6)$$

$$w^- = s + b; \quad w^+ = s + b - q_n(w^-, W^*). \quad (4.7)$$

At  $t = T$ , we have the terminal condition

$$V(s, b, W^*, T^+) = \kappa \left( W^* + \frac{\min((s + b - W^*), 0)}{\alpha} \right). \quad (4.8)$$

## 4.2.3 Conditional Expectations between Rebalancing Times

For  $t \in (t_{n-1}, t_n)$ , there are no cash flows, and all quantities are inflation-adjusted with discounting applied. Consequently, using the tower property, we obtain the following for  $0 < h < (t_n - t_{n-1})$ :

$$V(s, b, W^*, t) = \mathbb{E}[V(S(t+h), B(t+h), W^*, t+h) \mid S(t) = s, B(t) = b]; \quad t \in (t_{n-1}, t_n - h). \quad (4.9)$$

To compute this conditional expectation, we apply Itô's Lemma for jump processes (Tankov and Cont, 2009) using parametric models for stock and bond processes. Applying Itô's Lemma in Equation (4.9), along with Equations (3.1) and (3.4), leads to the following equation:

$$\begin{aligned} V_t + \frac{(\sigma^s)^2 s^2}{2} V_{ss} + (\mu^s - \lambda_\xi^s \gamma_\xi^s) s V_s + \lambda_\xi^s \int_{-\infty}^{+\infty} V(e^y s, b, t) f^s(y) dy + \frac{(\sigma^b)^2 b^2}{2} V_{bb} \\ + (\mu^b + \mu^b 1_{\{b < 0\}} - \lambda_\xi^b \gamma_\xi^b) b V_b + \lambda_\xi^b \int_{-\infty}^{+\infty} V(s, e^y b, t) f^b(y) dy \\ - (\lambda_\xi^s + \lambda_\xi^b) V + \rho_{sb} \sigma^s \sigma^b V_{sb} = 0, \quad s \geq 0. \end{aligned} \quad (4.10)$$

Here, the density functions  $f^s(y)$  and  $f^b(y)$  are as defined in Equation (3.2). In practical computations, this resulting partial integro-differential equation (PIDE) is solved using Fourier methods, as discussed in Forsyth and Labahn (2019).

#### 4.2.4 Equivalence with $PCEE_{t_0}(\kappa)$

Proceeding backward in time, the auxiliary function  $V(s, b, W^*, t_0^-)$  is determined at time zero. Problem ( $PCEE_{t_0}(\kappa)$ ) (2.18) is then solved using a final optimization step

$$J(s, b, t_0^-) = \sup_{W'} V(s, b, W', t_0^-). \quad (4.11)$$

The function  $V(s, b, W', t_0^-)$  serves as the auxiliary function at the start of the investment period and represents the final step (moving backward) in solving the dynamic programming formulation. To derive this, we start with Equation (4.8) and recursively compute backward in time. Finally, we obtain Equation (2.18) by interchanging the  $\sup_{W'}$  and  $\sup_{\mathcal{P}}$  in the final step.

This formulation (4.2–4.9) corresponds to the problem ( $PCEE_{t_0}(\kappa)$ ) (2.18). For further computational details, refer to Forsyth (2022).

### 4.3 Neural Network Formulation

As an alternative to the HJB framework, Chen et al. (2023) propose a neural network-based approach to solve the stochastic optimal control Problem (2.18), which possesses the following characteristics.

1. The NN framework is data driven, which does not require a parametric model for traded assets being explicitly specified. This avoids postulating stochastic processes, being modeled non-parametrically. In addition, this allows us to add auxiliary market signals/variables (although we do not exploit this idea in this work).
2. The NN framework circumvents the need for computing high-dimensional conditional expectations by directly solving for the control at all times through a single standard unconstrained optimization, bypassing dynamic programming (see Staden et al. (2023) for further discussion). Given that the control is low-dimensional, this approach mitigates the curse of dimensionality by determining the control directly rather than relying on value iteration, as in the HJB dynamic programming method, see Staden et al. (2023). Additionally, this method eliminates backward error propagation across rebalancing times.
3. If the optimal control varies continuously with time and state, the NN control will inherently capture this behavior. In cases where the optimal control is discontinuous, the NN representation provides a smooth approximation. Although continuity is not explicitly required by the original problem formulation in (2.18), it is likely to offer practical advantages in implementation.
4. The NN method is highly scalable and can be efficiently extended to problems with longer time horizons or higher rebalancing frequencies without substantially increasing computational complexity. This contrasts with existing methods that rely on a stacked neural network approach (Tsang and Wong, 2020).

We now formally introduce the proposed NN framework and illustrate its key properties. The control in  $\mathcal{P}$  is directly approximated using feed-forward, fully connected neural networks. Given the parameters  $\theta_p$  and  $\theta_q$ , which represent the NN weights and biases, the functions  $\hat{p}(W(t_i), t_i, \theta_p)$  and  $\hat{q}(W(t_i), t_i, \theta_q)$  serve as approximations for the controls  $p_i$  and  $q_i$ , respectively.

$$\begin{aligned} \hat{q}(W_i^-, t_i^-, \theta_q) &\simeq q_i(W_i^-), \quad i = 0, \dots, M \\ \hat{p}(W_i^+, t_i^+, \theta_p) &\simeq p_i(W_i^+), \quad i = 0, \dots, M - 1 \\ \hat{\mathcal{P}} &= \{(\hat{q}(\cdot), \hat{p}(\cdot))\} \simeq \mathcal{P} \end{aligned}$$

The functions  $\hat{p}$  and  $\hat{q}$  take time as an input, allowing us to use just two NN functions to approximate the control  $\mathcal{P}$  over time, rather than defining a separate NN at each rebalancing point. In this section, we explain how Problem (2.18) is addressed using this approximation and provide a detailed description of the NN architecture, including activation functions that enforce stochastic constraints.

### 4.3.1 Neural Network Optimization for $PCEE_{t_0}(\kappa)$

We start by formulating the NN optimization problem for the stochastic optimal control Problem (2.18). In Section 4.2.2 and Section 4.2.3, we established that the controls  $q_i$  and  $p_i$  depend solely on wealth, assuming stochastic processes (3.1), (3.4). Our objective is to determine the NN weights  $\theta_p$  and  $\theta_q$  by solving (2.18), where  $\hat{q}(W_i^-, t_i^-, \theta_q)$  and  $\hat{p}(W_i^+, t_i^+, \theta_p)$  serve as approximations of the feasible controls  $(q_i, p_i) \in \mathcal{Z}(W_i^-, W_i^+, t_i)$  for  $t_i \in \mathcal{T}$ . For any given set of controls  $\mathcal{P}$  and wealth level  $W'$ , we define the performance criterion  $V_{NN}$  as follows.

$$V_{NN}(\hat{\mathcal{P}}, W', s, b, t_0^-) = \mathbb{E}_{\hat{\mathcal{P}}_0}^{X_0^-, t_0^-} \left[ \sum_{i=0}^M \hat{q}_i + \kappa \left( W' + \frac{1}{\alpha} \min(W_T - W', 0) \right) + \epsilon W_T \middle| X(t_0^-) = (s, b) \right]$$

subject to

$$\begin{cases} (S_t, B_t) \text{ follow processes (3.1) and (3.4); } & t \notin \mathcal{T}, \\ W_i^+ = S_i^- + B_i^- - q_i, & X_i^+ = (S_i^+, B_i^+), \\ S_i^+ = \hat{p}_i(\cdot) W_i^+, & B_i^+ = (1 - \hat{p}_i(\cdot)) W_i^+, \\ (\hat{q}_i(\cdot), \hat{p}_i(\cdot)) \in \mathcal{Z}(W_i^-, W_i^+, t_i), \\ i = 0, \dots, M, & t_i \in \mathcal{T}. \end{cases} \quad (4.12)$$

The optimal value function  $J_{NN}$  (at  $t_0^-$ ) is then given by

$$J_{NN}(s, b, t_0^-) = \sup_{W'} \sup_{\hat{\mathcal{P}} \in \mathcal{A}} V_{NN}(\hat{\mathcal{P}}, W', s, b, t_0^-). \quad (4.13)$$

Next we describe the structure of the neural networks and feasibility encoding.

### 4.3.2 Neural Network Framework

We introduce two fully connected feed-forward neural networks for allocation and withdrawal, represented by  $\hat{p}$  and  $\hat{q}$ , respectively. These networks are parameterized by weight and bias vectors for allocation and withdrawal:  $\theta_p \in \mathbb{R}^{\nu_p}$  and  $\theta_q \in \mathbb{R}^{\nu_q}$ . The two networks may differ in activation functions, the number of hidden layers, and nodes per layer. Both networks take inputs in the form  $(W(t_i), t_i)$ , but the withdrawal NN  $\hat{q}$  uses the state variable before withdrawal,  $(W(t_i^-), t_i)$ , while the allocation NN  $\hat{p}$  takes the state variable after withdrawal,  $(W(t_i^+), t_i)$ .

To ensure the NN generates a feasible control as outlined in (4.19), we apply a modified sigmoid activation function to scale the output of the withdrawal NN  $\hat{q}$  in accordance with the constraints on the withdrawal amount  $q_i$  from the  $PCEE_{t_0}(\kappa)$  problem, as specified in Equation (2.7). This approach enables unconstrained optimization of the NN training parameters.

Assuming  $x \in [0, 1]$ , the function  $a + (b - a)x$  scales the output within the range  $[a, b]$ . We constrain the withdrawal  $\hat{q}$  to lie within  $[q_{\min}, q_{\max}]$ . Notably, the withdrawal range  $q_{\max} - q_{\min}$  depends on the wealth level  $W^-$ , as referenced in Equation (2.11). The specific range of permitted withdrawals is defined as follows:

$$\text{range} = \begin{cases} q_{\max} - q_{\min}, & \text{if } W_i^- > q_{\max} \\ W^- - q_{\min}, & \text{if } q_{\min} < W_i^- < q_{\max} \\ 0, & \text{if } W_i^- < q_{\min} \end{cases} \quad (4.14)$$

More concisely, we have the following mathematical expression:

$$\text{range} = \max(\min(q_{\max}, W^-) - q_{\min}, 0). \quad (4.15)$$

Let  $z \in \mathbb{R}$  represent the NN output before applying the final activation function in the output layer of  $\hat{q}$ . The value of  $z$  is determined by the input features, state, and time, before undergoing transformation through the activation function. The withdrawal is then expressed as follows:

$$\hat{q}(W^-, t, \theta_q) = q_{\min} + \text{range} \cdot \left( \frac{1}{1 + e^{-z}} \right). \quad (4.16)$$

$$= q_{\min} + \max(\min(q_{\max}, W^-) - q_{\min}, 0) \cdot \left( \frac{1}{1 + e^{-z}} \right). \quad (4.17)$$

Note that the sigmoid function  $\frac{1}{1+e^{-z}}$  is a mapping from  $\mathbb{R} \rightarrow [0, 1]$ .

Similarly, we use a softmax activation function on the NN output for  $\hat{p}$ , ensuring no-short and no-leverage constraints are automatically satisfied.

With these output activation functions, it can be readily verified that  $(\hat{q}_i(\cdot), \hat{p}_i(\cdot))$  always belong to  $\mathcal{Z}(W_i^-, W_i^+, t_i)$ . Utilizing the defined NN, the problem (4.13) of determining the optimal  $\hat{\mathcal{P}}$  is reformulated as an optimization problem.

$$\hat{J}_{NN}(s, b, t_0^-) = \sup_{W' \in \mathbb{R}} \sup_{\theta_q \in \mathbb{R}^{\nu_q}} \sup_{\theta_p \in \mathbb{R}^{\nu_p}} \hat{V}_{NN}(\theta_q, \theta_p, W', s, b, t_0^-). \quad (4.18)$$

It is important to note that while the original control  $\mathcal{P}$  is constrained in (2.9), the formulation in (4.18) transforms it into an unconstrained optimization over  $\theta_q$ ,  $\theta_p$ , and  $W'$ . Consequently, Problem (4.18) can be solved directly using a gradient descent approach (Kingma and Ba, 2014). In the numerical experiments detailed in Section 5, we employ the Adam stochastic gradient descent optimizer to determine the optimal values of  $\theta_q^*$ ,  $\theta_p^*$ , and  $W'$ .

Figure 4.1 illustrates the NN framework. Note that the output of NN  $\hat{q}$  yields the amount to withdraw, while the output of NN  $\hat{p}$  produces asset percentage allocations.

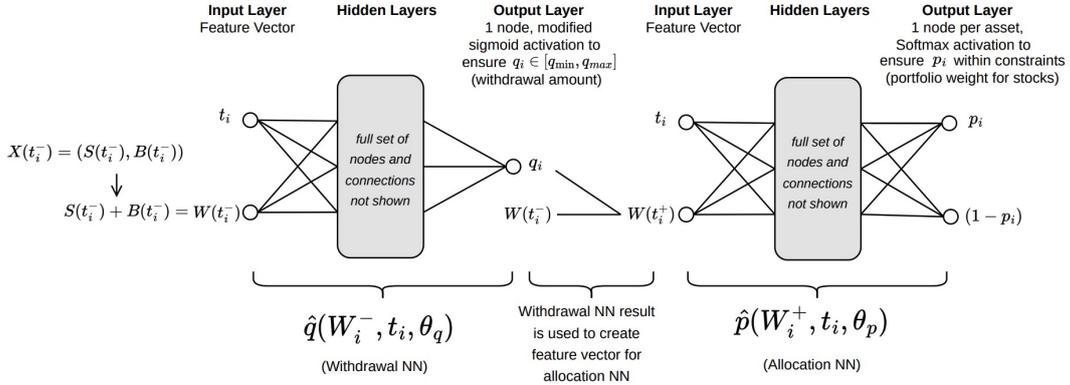


Figure 4.1: Illustration of the NN framework as per Section 4.3. (Chen et al., 2023)

We emphasize the key aspects of this NN structure:

- Time serves as an *input* to both neural networks in the framework. The parameter vectors  $\theta_q$  and  $\theta_p$  remain constant and do not change over time.
- At each rebalancing instance, the observed wealth before withdrawal is used to construct the feature vector for  $\hat{q}$ . The resulting withdrawal is then utilized to compute the post-withdrawal wealth, which serves as an input feature for  $\hat{p}$ .
- Standard sigmoid activation functions are applied at each *hidden layer* output.
- The activation functions for withdrawal and allocation differ. The control  $\hat{q}$  employs a modified sigmoid activation function to transform its output according to the constraints in (2.7). Meanwhile,  $\hat{p}$  utilizes a softmax activation to ensure that portfolio weights remain positive and sum to one, as specified in (2.18).

- By incorporating these activation functions to constrain the NN output, we enable unconstrained optimization for training the neural network.

### 4.3.3 NN Estimate of the Optimal Control

Now we describe the NN training optimization problem for the decumulation problem, based on return sample trajectories. We assume that a set of asset return trajectories are available, which are used to approximate the expectation in (4.1) for any given control. For NN training, we approximate the expectation in (4.1) based on a finite number of samples as follows:

$$\hat{V}_{NN}(\boldsymbol{\theta}_q, \boldsymbol{\theta}_p, W', s, b, t_0^-) = \frac{1}{N} \sum_{j=1}^N \left[ \sum_{i=0}^M \hat{q}((W_i^-)^j, t_i; \boldsymbol{\theta}_q) + \kappa \left( W' + \frac{1}{\alpha} \min((W_T)^j - W', 0) \right) + \epsilon(W_T)^j \middle| X(t_0^-) = (s, b) \right] \quad (4.19)$$

$$\text{subject to } \begin{cases} ((S_t)^j, (B_t)^j) \text{ is the } j^{\text{th}} \text{ sample of values, } t \notin \mathcal{T}, \\ (W_i^+)^j = (S_i^-)^j + (B_i^-)^j - \hat{q}((W_i^-)^j, t_i, \boldsymbol{\theta}_q), \quad (X_i^+)^j = (S_i^+, B_i^+)^j, \\ (S_i^+)^j = \hat{p}((W_i^+)^j, t_i, \boldsymbol{\theta}_p)(W_i^+)^j, \quad (B_i^+)^j = (1 - \hat{p}((W_i^+)^j, t_i, \boldsymbol{\theta}_p))(W_i^+)^j, \\ (\hat{q}_i(\cdot), \hat{p}_i(\cdot)) \in \mathcal{Z}((W_i^-)^j, (W_i^+)^j, t_i), \\ i = 0, \dots, M, \quad t_i \in \mathcal{T}. \end{cases}$$

The superscript  $j$  denotes the  $j^{\text{th}}$  path of joint asset returns, while  $N$  represents the total number of sampled paths. For benchmark comparison, we generate price paths using the processes in (3.1) and (3.4). However, any method can be employed to generate these paths, as we are not limited to parametric SDEs. We assume that random sample paths are independent, although correlations may exist between the returns of different assets. Additionally, correlations between returns from different time periods can also be incorporated. For instance, block bootstrap resampling is specifically designed to capture autocorrelation in time series data.

The optimal parameters obtained from training the neural network are used to define the control functions  $\hat{q}^*(\cdot) := \hat{q}(\cdot; \boldsymbol{\theta}_q^*)$  and  $\hat{p}^*(\cdot) := \hat{p}(\cdot; \boldsymbol{\theta}_p^*)$ , respectively. These functions enable us to assess the performance of the generated control on test datasets, including both out-of-sample and out-of-distribution scenarios. The detailed results of these evaluations are presented in Chapter 6.

### 4.3.4 Stochastic Gradient Method

Stochastic Gradient Method (SGM), also known as Stochastic Gradient Descent (SGD), is a fundamental optimization algorithm widely used in training neural networks and solving stochastic optimal control problems. The Stochastic Gradient Method is used to optimize the parameters of the neural networks that approximate the control functions for withdrawal  $q_i$  and allocation  $p_i$ .

The key steps in the Stochastic Gradient Method for NN framework (4.19) are as follows:

1. **Parameter Initialization:** The neural network parameters  $\boldsymbol{\theta}_q$  and  $\boldsymbol{\theta}_p$  (representing weights and biases) are initialized randomly or using a predefined strategy.
2. **Gradient Estimation:** For each iteration, a mini-batch of sample paths is drawn from the available data. The gradient of the objective function with respect to the NN parameters is estimated using these samples. The objective function, as defined in Equation (4.19), is:

$$\hat{V}_{NN}(\boldsymbol{\theta}_q, \boldsymbol{\theta}_p, W', s, b, t_0^-) = \frac{1}{N} \sum_{j=1}^N \left[ \sum_{i=0}^M \hat{q}((W_i^-)^j, t_i; \boldsymbol{\theta}_q) + \kappa \left( W' + \frac{1}{\alpha} \min((W_T)^j - W', 0) \right) + \epsilon (W_T)^j \mathbb{1}_{X(t_0^-) = (s, b)} \right] \quad (4.20)$$

where  $N$  is the number of sample paths, and  $(W_i^-)^j$  and  $(W_T)^j$  represent the wealth before withdrawal at time  $t_i$  and terminal wealth for the  $j$ -th path, respectively.

3. **Parameter Update:** The parameters  $\boldsymbol{\theta}_q$  and  $\boldsymbol{\theta}_p$  are updated using the estimated gradient. The update rule for SGD is:

$$\boldsymbol{\theta}_{q,p} \leftarrow \boldsymbol{\theta}_{q,p} - \eta \nabla_{\boldsymbol{\theta}_{q,p}} \hat{V}_{NN}, \quad (4.21)$$

where  $\eta$  is the learning rate, which controls the step size of the update. The learning rate can be adjusted dynamically using techniques like learning rate scheduling or adaptive methods like Adam (Adaptive Moment Estimation).

4. **Iteration:** Steps 2 and 3 are repeated until the parameters converge to a solution that maximizes the objective function. In our Optimal Control Problem, we manually set the number of iterations instead of relying on an automatic stopping criterion. Hyper-parameter settings will be provided in Table 5.2.

For the Stochastic Gradient Method iteration process, several key factors (e.g. [learning rate](#), [learning rate decaying schedules](#), [batch size](#)) can significantly impact the training performance. These factors are modified in the NN training experiments in Section 5.2 to assess their impact on the computational results.

## Learning Rate

The learning rate  $\eta$  in equation (4.21) is one of the most critical hyper-parameters in deep learning optimization, directly influencing the convergence speed, stability, and final performance of the trained model. It determines the step size at which the model updates its parameters in response to the computed gradients. A well-chosen learning rate can lead to fast convergence and optimal generalization, while an improper learning rate—either too high or too low—can severely impact training dynamics.

A large learning rate allows the optimizer to make big updates to model parameters, accelerating convergence during the early stages of training. However, if  $\eta$  is too large, it can lead to unstable training, causing the model to oscillate around the optimal solution or even diverge, failing to converge altogether. This occurs because large step sizes may overshoot the optimal region, preventing the optimizer from settling into a minimum of the loss function (Ruder, 2016).

Conversely, a small learning rate ensures that the model makes small, incremental updates, leading to more stable training and finer adjustments to the model parameters. This is particularly useful when objective function is very non-linear and in the later stages of training when the optimizer is expected to refine the learned features. However, if  $\eta$  is too small, training can become excessively slow, requiring a prohibitively large number of iterations to reach an optimal solution (Ruder, 2016). Moreover, a small learning rate can cause the model to get stuck in sharp local minima<sup>1</sup> in the loss surface, where the gradients are too small to facilitate meaningful updates. This can lead to the model fails to optimize the objective function.

To balance these trade-offs, modern deep learning optimization techniques employ adaptive learning rates and learning rate decay schedules. Adaptive optimizers, such as Adam and RM-Sprop, dynamically adjust the learning rate based on past gradient information, allowing the

<sup>1</sup>Sharp minima refer to points in the loss landscape where the loss function has a high curvature, meaning that small changes in model parameters result in significant fluctuations in the loss.

model to take larger steps in regions of high variance and smaller steps in stable regions (Kingma and Ba, 2014). Meanwhile, learning rate decay schedules, such as multistep decay, cyclic learning rates, exponential decay, and cosine annealing, progressively reduce the learning rate as training progresses (Ruder, 2016; Smith, 2017; Johnson et al., 2023). This strategy helps the model take large steps early in training for faster convergence while allowing smaller, more precise updates later to fine-tune the parameters.

## Learning Rate Decaying Schedules

A learning rate decaying schedule is a technique used in neural network optimization to systematically adjust the learning rate during training. A well-crafted learning rate decay schedule seeks to strike a balance between the advantages of large and small learning rates. There are several common approaches to learning rate decay:

1. **Multistep Decay** Multistep decay is a learning rate scheduling strategy that reduces the learning rate at predefined training milestones. A milestone refers to a specific point during training (often measured as a percentage of total iterations) where the learning rate is adjusted. At each milestone, the learning rate is multiplied by a decay factor  $\gamma$ , following the formula:

$$\eta_a = \eta_0 \times \gamma^a \quad (4.22)$$

where:

- $\eta_a$  is the learning rate after  $a$  milestones,
- $\eta_0$  is the initial learning rate,
- $\gamma$  is the decay factor (typically between 0 and 1),
- $a$  is the number of milestones passed.

In our implementation, milestones are set at 70% and 97% of the training process, meaning the learning rate is reduced at these specific points to refine the model’s learning. The decay factor used is  $\gamma = 0.2$ , meaning that each time a milestone is reached, the learning rate is scaled down by 20% of its previous value.

2. **Cyclic Learning Rate** Cyclic learning rate (Smith, 2017) is a learning rate scheduling strategy that oscillates the learning rate between a predefined lower and upper bound. Instead of monotonically decreasing the learning rate, CLR alternates it in a cyclic manner to allow exploration of different regions in the loss landscape. The learning rate follows a triangular function:

$$\eta_b = \eta_{\min} + (\eta_{\max} - \eta_{\min}) \times \frac{|b \bmod (2HC) - HC|}{HC} \quad (4.23)$$

where:

- $\eta_b$  is the learning rate after  $b$  iterations,
- $\eta_{\min}$  and  $\eta_{\max}$  are the minimum and maximum learning rates, respectively,
- $HC$  is the half-cycle length, determining how frequently the learning rate oscillates.

In our implementation,  $HC = 2000$ , with the learning rate oscillating between  $\eta_{\min} = 0.01 \times \eta_0$  and  $\eta_{\max} = \eta_0$  ( $\eta_0$  is the initial learning rate).

3. **Exponential Decay** Exponential decay is a learning rate scheduling strategy that continuously reduces the learning rate over time. It applies a multiplicative decay factor at each iteration, which progressively decreases the learning rate. The learning rate follows the exponential decay function:

$$\eta_c = \eta_0 \times e^{-\beta c} \quad (4.24)$$

where:

- $\eta_c$  is the learning rate after  $c$  iterations,
- $\eta_0$  is the initial learning rate,
- $\beta$  is the decay rate, controlling the rate of reduction.

In our implementation, the decay factor is set to  $\alpha = e^{-\beta} = 0.99991$ .

4. **Cosine Annealing** Cosine annealing (Loshchilov and Hutter, 2016) is a learning rate scheduling strategy that gradually adjusts the learning rate following a cosine function. At each iteration, the learning rate is adjusted according to the cosine function, following the formula:

$$\eta_d = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min}) \left( 1 + \cos \left( \frac{\pi d}{P} \right) \right) \quad (4.25)$$

where:

- $\eta_d$  is the learning rate after  $d$  iterations,
- $\eta_{\min}$  and  $\eta_{\max}$  are the minimum and maximum learning rates, respectively,
- $P$  is the total decay period, which determines how long it takes for the learning rate to decrease from  $\eta_{\max}$  to  $\eta_{\min}$ .  $\eta_{\max} = \eta_0, \eta_{\min} = 0$ .

In our implementation, the decay period is set to  $P = 5000$  iterations, meaning the learning rate follows a cosine curve from  $\eta_{\max}$  to  $\eta_{\min}$  over 5000 iterations.

### Batch Size

Batch size is a fundamental hyper-parameter in deep learning optimization. It defines the number of training samples used per iteration before the model updates its parameters.

Large batch training processes more samples per iteration, improving computational efficiency and stabilizing gradient updates. It enables faster convergence but often leads to poor generalization due to convergence to sharp minima (Keskar et al., 2016). Small batch training introduces higher gradient variance, which enhances generalization by preventing convergence to sharp minima (Smith et al., 2020).

# Chapter 5

## Computational Investigation

Building upon the work of [Chen et al. \(2023\)](#), this thesis conducts a deeper investigation into the robustness and efficiency of neural network-based stochastic optimal control. Specifically, we explore the performance of the model under high-risk aversion scenarios, comparing single transfer learning with sequential transfer learning to determine the most effective method for improving training stability at high-risk aversion levels. To further validate the model's performance, we compare a heat map to examine the optimal control outcomes derived from HJB-based methods with those obtained using neural networks. Finally, we incorporate updated historical data and expand the asset universe, evaluating the model's performance using multi-asset training datasets, long-term log-returns, and computational results based on CRSP asset data. Through these extensions, we aim to refine the NN-based optimization framework and provide deeper insights into its advantages over traditional approaches in dynamic asset allocation.

### 5.1 Investment Scenario and NN Parameter Settings

Tables [5.1](#) and [5.2](#) provide a comprehensive overview of the investment scenario in our study and neural network (NN) parameter settings used in [Chen et al. \(2023\)](#). Table [5.1](#) outlines the key assumptions and constraints for the investment problem, including the investment horizon, asset types, withdrawal constraints, and portfolio rebalancing rules. These settings define the financial environment within which the optimization model operates. Table [5.2](#) details the hyper-parameters used in training the neural network framework, including the network architecture, optimization settings, and learning rate schedules. These parameters are carefully chosen to ensure efficient training, stable convergence, and effective learning of the optimal withdrawal and allocation strategies.

Investment horizon $T$ (years)	30
Stock index	CPI adjusted CRSP Value-Weighted Index
Bond index	CPI adjusted CRSP 10-Year T-Bond Returns
Initial portfolio value $W_0$	1000
Cash withdrawal times	$t = 0, 1, \dots, 30$
Withdrawal range	[35, 60]
Equity fraction range	[0, 1]
Borrowing spread $\mu_c^b$	0.0
Rebalancing interval (years)	1
Market model parameters	See Chapter 3

Table 5.1: *Problem setup and input data. Monetary units: USD\$ in thousands.*

NN framework hyper-parameter	Value
Hidden layers per network	2
# of nodes per hidden layer	10
Nodes have biases	True
# of iterations (#itn)	50,000
SGD mini-batch size	1,000
# of training paths	$2.56 \times 10^5$
Optimizer	Adaptive Momentum
Initial Adam learning rate for $(\theta_q, \theta_p)$	0.05
Initial Adam learning rate for $W'$	0.04
Adam learning rate decay schedule	$[0.70 \times \text{\#itn}, 0.97 \times \text{\#itn}]$ , $\gamma = 0.20$
Adam $\beta_1$	0.9
Adam $\beta_2$	0.998
Adam weight decay ( $\ell_2$ Penalty)	0.0001

Table 5.2: *Hyper-parameters used in training the NN framework for numerical experiments.* (Chen et al., 2023)

In several of our experiments (Tables 5.6, 5.7, 5.8, 5.9, 5.10, and 5.11), we ensure the consistency of our results by introducing controlled randomness into the Stochastic Gradient Descent (SGD) process. Specifically, we vary the randomness by applying different random seeds in Python’s NumPy library. This approach affects both the parameter initializations and the selection of mini-batches of sample paths (as detailed in Section 4.3.4), while keeping the original dataset of  $2.56 \times 10^5$  observations of data unchanged. This methodology allows us to evaluate the robustness and reproducibility of our results under varying initial conditions and training dynamics.

## 5.2 Emphasis on Minimizing Risk

When individuals prioritize risk minimization, the objective function (2.18) suggests that a larger  $\kappa$  is needed, as it places greater emphasis on controlling downside risk while balancing expected withdrawals. In this section, we focus on exploring the model with  $\kappa = 50$ , which represents a context where risk aversion is significantly high. By analyzing this setting in depth, we aim to evaluate how the model adjusts withdrawal and allocation strategies under extreme risk-averse conditions and assess the stability of the learned strategies compared to those obtained from HJB-based solutions.

### 5.2.1 Computational Challenges in Minimizing Risk

In the context of this thesis, which focuses on optimizing withdrawal and asset allocation strategies using a neural network-based framework for dynamic portfolio management, minimizing only risk presents significant computational and theoretical challenges. When the objective function is primarily weighted towards optimizing Expected Shortfall (ES), i.e., at higher values of the risk aversion parameter  $\kappa$  in equation (2.18), the training process becomes substantially more difficult. This is because the ES measure (Conditional Value at Risk, CVaR) is only influenced by the lower tail (typically the worst 5%) of the terminal wealth distribution. Since tail-end data samples are inherently sparse, the model struggles to obtain sufficient gradient updates in minimizing risk, leading to increased difficulty in convergence.

In Chen et al. (2023), this issue was addressed by employing transfer learning, where models were first trained with low  $\kappa$  values and then progressively refined to achieve high  $\kappa$  solutions. Instead of relying on this sequential approach, we explore whether it is possible to directly train a model with a high-risk aversion setting by adjusting other hyper-parameters, such as initial learning rate, learning rate schedules, batch size. This investigation aims to determine whether

an alternative optimization strategy can achieve stable convergence without requiring a gradual transfer learning framework.

Training models with high risk aversion  $\kappa = 50.0$  presents significant challenges due to the sample sparsity of tail-end data and the heightened sensitivity to extreme risk scenarios. We conjecture that to address these issues, two key adjustments may be effective: reducing the learning rate to enhance convergence stability and increasing the batch size to ensure sufficient exposure to tail samples.

## 5.2.2 Computational Results

We first validate the increased difficulty of directly training models with high  $\kappa$  values by comparing the training results of models with low and high  $\kappa$ , using the Hamilton-Jacobi-Bellman (HJB) solution as a benchmark. Table 5.3 shows that low  $\kappa$  models closely approximate the HJB solution, while high  $\kappa$  models exhibit significant deviations. This discrepancy indicates that training high  $\kappa$  models is more challenging, likely due to the sparsity of tail-end data in the CVaR-based objective function, which increases sensitivity to extreme risk scenarios.

By comparing the differences between model outputs and the HJB benchmark, this study provides empirical evidence that direct training of high  $\kappa$  models is more difficult.

Figure 5.1 illustrates the increased difficulty of training high  $\kappa$  models compared to low  $\kappa$  models. The NN results for  $\kappa = 0.05$  closely align with the HJB solution, indicating successful training performance. However, for  $\kappa = 50$ , the NN model significantly deviates from the HJB benchmark, failing to approximate the optimal control. This suggests that directly training high  $\kappa$  models is more challenging due to the sparsity of tail-end data, leading to unstable optimization and poor convergence.

	$\kappa$ value	learning rate schedule	NN obj val	HJB obj val	% difference
exp 1	0.05	multistep	1741.73	1741.54	0.01%
exp 2	0.05	cyclic	1741.75	1741.54	0.01%
exp 3	0.05	exponential	1737.83	1741.54	-0.21%
exp 4	0.05	cosine annealing	1741.69	1741.54	0.01%
exp 5	50.0	multistep	1659.51	2946.70	-43.68%
exp 6	50.0	cyclic	2575.75	2946.70	-12.59%
exp 7	50.0	exponential	2073.66	2946.70	-29.63%
exp 8	50.0	cosine annealing	2568.74	2946.70	-12.83%

Table 5.3: Comparison of direct training results for the  $\kappa = 0.05$  model and  $\kappa = 50$  model by Neural Network (NN) method, with controls computed from Problem (2.18). Investment setup in Table 5.1. Hyper-parameters setup in Table 5.2 with varying learning rate decay schedules. Model was trained on  $2.56 \times 10^5$  observations of simulated data from calibrated jump models.

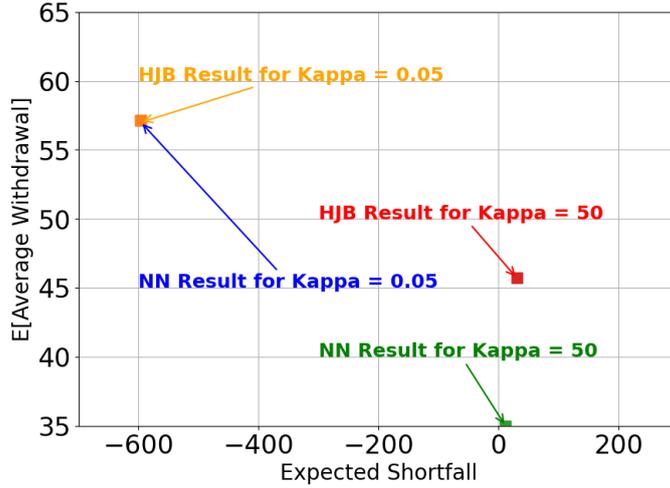


Figure 5.1: Comparison of direct training results for the  $\kappa = 0.05$  model and  $\kappa = 50$  model by Neural Network (NN) method, with controls computed from Problem (2.18). Investment setup in Table 5.1. Hyper-parameters setup in Table 5.2. Model was trained on  $2.56 \times 10^5$  observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12.

Then, we further explore the performance on direct training with high-risk aversion situations by adjusting the hyper-parameters according to the description in Section 5.2.1. See performance in Table 5.4 and Table 5.5.

The results in Tables 5.4 and 5.5 demonstrate that even with careful adjustments to learning rate, and batch size, directly training high  $\kappa$  models ( $\kappa = 50$ ) fails to achieve results that closely match the Hamilton-Jacobi-Bellman (HJB) benchmark. While lower learning rates improve training stability and larger batch sizes ensure better inclusion of tail-end data samples, the final NN objective values remain significantly different from the HJB solution, with percentage difference exceeding 10% in all cases.

	lr schedule	NN lr	$W'$ lr	itbound	batch	NN obj val	HJB obj val	% difference
exp 1	multistep	0.05	0.04	50000	1000	1659.51	2946.70	-43.68%
exp 2	cyclic	0.05	0.04	50000	1000	2575.75	2946.70	-12.59%
exp 3	multistep	0.04	0.03	50000	1000	2579.28	2946.70	-12.47%
exp 4	cyclic	0.04	0.03	50000	1000	2566.72	2946.70	-12.90%
exp 5	multistep	0.03	0.02	50000	1000	1878.44	2946.70	-36.25%
exp 6	cyclic	0.03	0.02	50000	1000	2577.12	2946.70	-12.54%
exp 7	multistep	0.02	0.01	50000	1000	2582.63	2946.70	-12.36%
exp 8	cyclic	0.02	0.01	50000	1000	2570.82	2946.70	-12.76%
exp 9	multistep	0.01	0.005	50000	1000	2565.91	2946.70	-12.92%
exp 10	cyclic	0.01	0.005	50000	1000	2503.64	2946.70	-15.04%
exp 11	multistep	0.005	0.001	50000	1000	1920.11	2946.70	-34.84%
exp 12	cyclic	0.005	0.001	50000	1000	1711.47	2946.70	-41.92%
exp 13	multistep	0.005	0.004	100000	1000	2546.34	2946.70	-13.59%
exp 14	cyclic	0.005	0.004	100000	1000	2556.99	2946.70	-13.23%
exp 15	multistep	0.001	0.001	500000	1000	1646.93	2946.70	-44.11%
exp 16	cyclic	0.001	0.001	500000	1000	2557.91	2946.70	-13.19%
exp 17	multistep	0.005	0.004	500000	100	2573.07	2946.70	-12.68%
exp 18	cyclic	0.005	0.004	500000	100	2568.82	2946.70	-12.82%

Table 5.4: Comparison of direct training results for the  $\kappa = 50$  model by Neural Network (NN) method with varying hyper-parameter settings, with controls computed from Problem (2.18). Investment setup in Table 5.1. Hyper-parameters setup in Table 5.2 with decreasing initial learning rates. Model was trained on  $2.56 \times 10^5$  observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12.

	batch size	lr schedule	NN obj val	HJB obj val	% difference
exp 1	5000	multistep	2634.56	2946.70	-10.59%
exp 2	5000	cyclic	2408.97	2946.70	-18.25%
exp 3	5000	exponential	2588.28	2946.70	-12.16%
exp 4	6000	multistep	2648.08	2946.70	-10.13%
exp 5	6000	cyclic	2595.98	2946.70	-11.90%
exp 6	6000	exponential	2591.60	2946.70	-12.05%
exp 7	7000	multistep	2615.79	2946.70	-11.23%
exp 8	7000	cyclic	2587.24	2946.70	-12.20%
exp 9	7000	exponential	2592.30	2946.70	-12.03%
exp 10	8000	multistep	2613.16	2946.70	-11.32%
exp 11	8000	cyclic	2592.21	2946.70	-12.03%
exp 12	8000	exponential	2580.80	2946.70	-12.42%
exp 13	9000	multistep	2605.18	2946.70	-11.59%
exp 14	9000	cyclic	2593.89	2946.70	-11.97%
exp 15	9000	exponential	2582.92	2946.70	-12.35%
exp 16	10000	multistep	2420.43	2946.70	-17.86%
exp 17	10000	cyclic	2595.39	2946.70	-11.92%
exp 18	10000	exponential	2282.38	2946.70	-22.54%

Table 5.5: Comparison of direct training results for the  $\kappa = 50$  model by Neural Network (NN) method with varying hyper-parameter settings, with controls computed from Problem (2.18). Investment setup in Table 5.1. Hyper-parameters setup in Table 5.2 with increasing batch size. Model was trained on  $2.56 \times 10^5$  observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12.

These findings suggest that the optimization landscape for high  $\kappa$  models is inherently challenging. Despite tuning hyper-parameters, the training process struggles to converge to the optimal control.

### 5.2.3 Transfer Learning

Given the difficulties in training low-risk models (high- $\kappa$  models), transfer learning becomes a necessary approach. Transfer learning is a machine learning technique that leverages knowledge from a pre-trained model to improve learning in a target task with limited data or challenging optimization landscapes. Instead of training a model from scratch, a model trained on a related problem serves as an initialization, allowing for faster convergence in training (Zhuang et al., 2021).

Mathematically, let  $\mathcal{D}_{source} = \{(x_m^{source}, y_m^{source})\}$  represent the source domain dataset and  $\mathcal{D}_{target} = \{(x_m^{target}, y_m^{target})\}$  represent the target domain dataset, where  $m$  indexes the individual data samples in the respective datasets. The goal of transfer learning is to transfer knowledge from a model trained on  $\mathcal{D}_{source}$  to improve the learning of a function  $f_{target}(x)$  in  $\mathcal{D}_{target}$ . If  $\theta_{source}$  represents the learned parameters in the source model, instead of randomly initializing  $\theta_{target}$  for the target model, we set:

$$\theta_{target}^0 = \theta_{source} \quad (5.1)$$

where  $\theta_{target}^0$  is the initial parameter state of the target model.

$$\theta_{target}^{(m+1)} = \theta_{target}^{(m)} - \eta \nabla \mathcal{L}_{target}(\theta_{target}^{(m)}) \quad (5.2)$$

where  $\mathcal{L}_{target}$  is the loss function for the target task, and  $\eta$  is the learning rate, and  $m$  denotes the iteration index in the training process. This approach ensures that the model retains useful features from the source task while gradually adapting to the new objective.

In the context of high  $\kappa$  models, transfer learning allows us to first train a low  $\kappa$  model, which is easier to optimize, and then use its parameters as initialization for the high  $\kappa$  model. This helps overcome the sparsity of tail-end data samples, making it possible to obtain a well-optimized solution.

## Single Transfer Learning

Since direct training the  $\kappa = 50$  model fails to achieve results comparable to the HJB benchmark, transfer learning is introduced to improve training convergence. Among all  $\kappa$  models, where  $\kappa$  takes values from the set  $[0.05, 0.2, 0.5, 1.0, 1.5, 3.0, 5.0, 50.0]$ , the  $\kappa = 0.05$  model consistently demonstrates the most stable and accurate training outcomes, closely approximating the HJB solution. Therefore, we choose the  $\kappa = 0.05$  model as the pre-trained source model for single transfer learning, using its trained parameters to initialize each higher  $\kappa$  model separately. This approach leverages the well-conditioned optimization landscape of low  $\kappa$  models to facilitate learning for high  $\kappa$  models, mitigating issues related to data sparsity. Table 5.6 shows the objective function value performances of single transfer learning when using  $\kappa = 0.05$  model as the source model.

$\kappa$	exp 1	exp 2	exp 3	exp 4	exp 5	NN avg obj val	HJB obj val	% difference
0.05	1741.77	1741.76	1741.75	1741.77	1741.77	1741.76	1741.54	0.01%
0.2	1673.77	1673.68	1673.78	1673.77	1673.82	1673.76	1674.41	-0.04%
0.5	1606.84	1605.77	1606.92	1606.57	1606.86	1606.59	1607.26	-0.04%
1.0	1563.21	1561.60	1562.40	1561.13	1562.94	1562.26	1568.45	-0.39%
1.5	1527.21	1528.42	1546.29	1527.35	1526.12	1531.08	1557.46	-1.69%
3.0	1564.77	1562.88	1441.74	1416.24	1442.58	1485.64	1569.71	-5.36%
5.0	1605.49	1603.92	1334.62	1594.05	1604.64	1548.55	1612.16	-3.95%
50.0	2907.14	2905.40	2910.63	2615.62	2818.51	2831.46	2946.70	-3.91%

Table 5.6: *Separate single transfer learning results for training each  $\kappa$  model, with controls computed from Problem (2.18). The  $\kappa = 0.05$  model is used as the source model for transfer learning to other  $\kappa$  models. Model was trained on  $2.56 \times 10^5$  observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12. The investment setup is detailed in Table 5.1, and the hyper-parameter configuration is provided in Table 5.2. Each model is trained five times with different random seeds (0, 1, 2, 3, 4) to examine consistency across runs, refer to Section 5.1.*

From Figures 5.2 - 5.8, when applying single transfer learning directly from the  $\kappa = 0.05$  model to other higher- $\kappa$  models, the resulting models tend to exhibit both higher expected withdrawal (higher reward) and lower expected shortfall (higher risk) compared to the HJB benchmark. This suggests that the transferred initialization inherits characteristics of the low- $\kappa$  model, which emphasizes higher reward feature. However, for some intermediate  $\kappa$  values (i.e.,  $\kappa = 1.0, 1.5, 3.0$ ), applying transfer learning from the  $\kappa = 0.05$  model results in worse performance compared to training these models without transfer learning, leading to a greater deviation from the HJB benchmark. Nevertheless, for  $\kappa = 5.0$  and  $\kappa = 50.0$ , transfer learning proves highly beneficial. These models, which are difficult to train directly due to the sparsity of tail-end data, benefit significantly from the initialization provided by the  $\kappa = 0.05$  model.

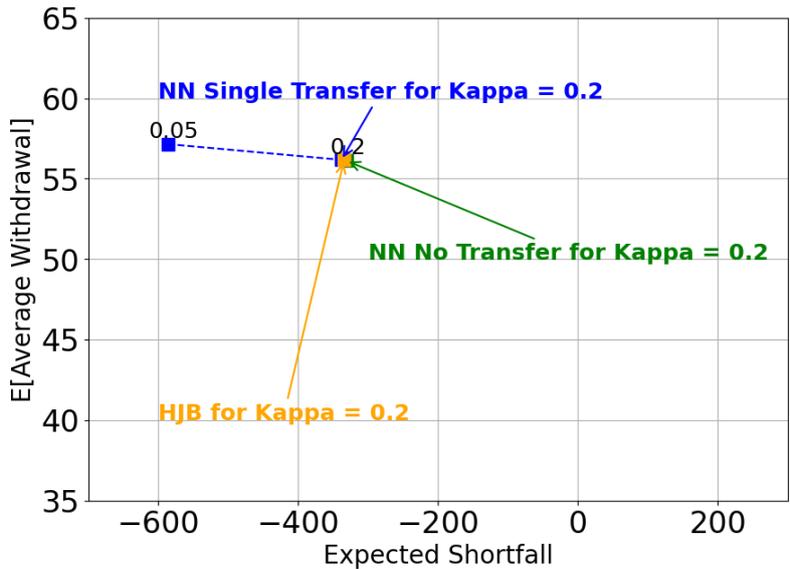


Figure 5.2: Comparison of (EW, ES) outcomes for training  $\kappa = 0.2$  model, with controls computed from Problem (2.18). The  $\kappa = 0.05$  model is used as the source model for transfer learning to  $\kappa = 0.2$  models. Model was trained on  $2.56 \times 10^5$  observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12. The investment setup is detailed in Table 5.1, and the hyper-parameter configuration is provided in Table 5.2.

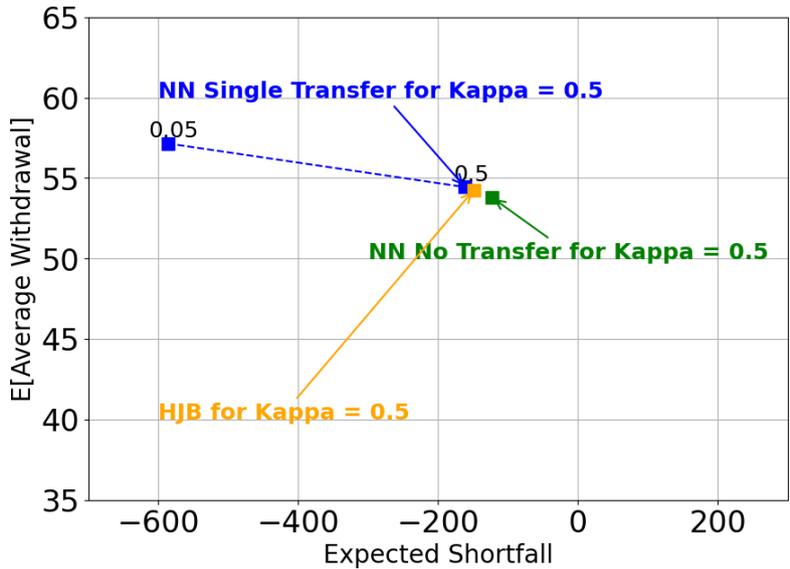


Figure 5.3: Comparison of (EW, ES) outcomes for training  $\kappa = 0.5$  model, with controls computed from Problem (2.18). The  $\kappa = 0.05$  model is used as the source model for transfer learning to  $\kappa = 0.5$  models. Model was trained on  $2.56 \times 10^5$  observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12. The investment setup is detailed in Table 5.1, and the hyper-parameter configuration is provided in Table 5.2.

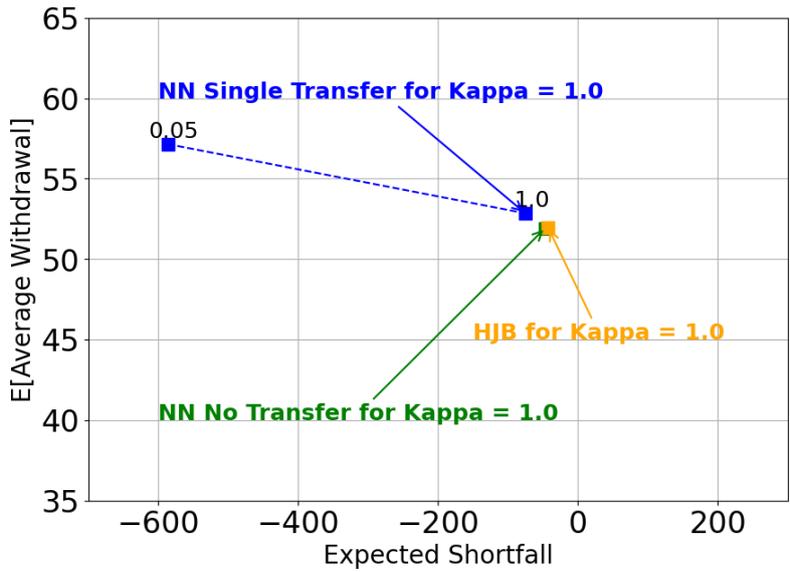


Figure 5.4: Comparison of (EW, ES) outcomes for training  $\kappa = 1.0$  model, with controls computed from Problem (2.18). The  $\kappa = 0.05$  model is used as the source model for transfer learning to  $\kappa = 1.0$  models. Model was trained on  $2.56 \times 10^5$  observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12. The investment setup is detailed in Table 5.1, and the hyper-parameter configuration is provided in Table 5.2.

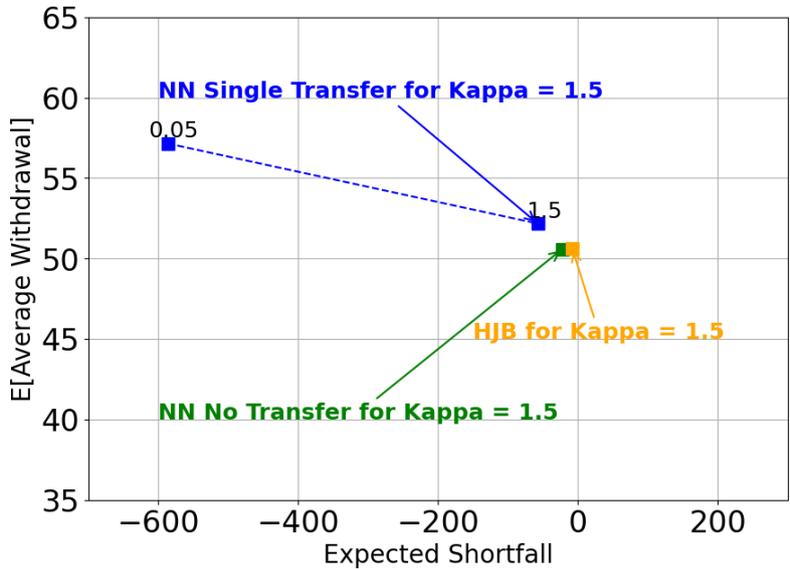


Figure 5.5: Comparison of (EW, ES) outcomes for training  $\kappa = 1.5$  model, with controls computed from Problem (2.18). The  $\kappa = 0.05$  model is used as the source model for transfer learning to  $\kappa = 1.5$  models. Model was trained on  $2.56 \times 10^5$  observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12. The investment setup is detailed in Table 5.1, and the hyper-parameter configuration is provided in Table 5.2.

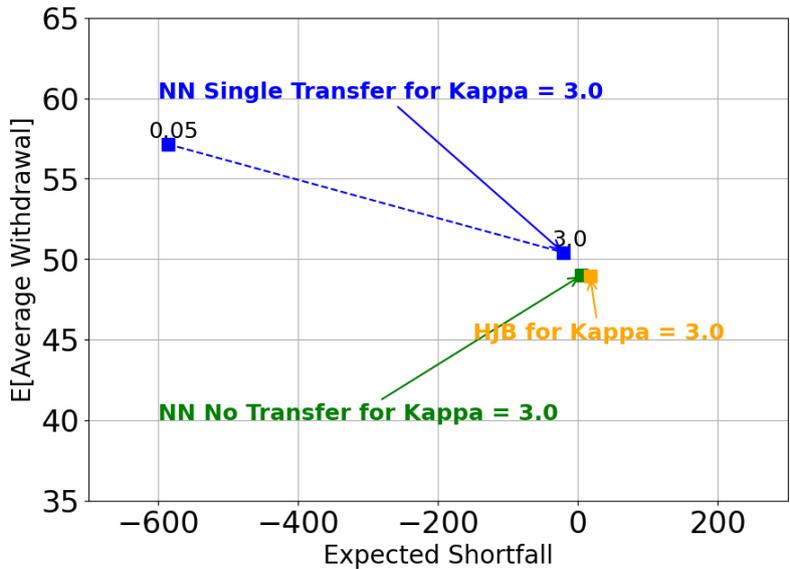


Figure 5.6: Comparison of (EW, ES) outcomes for training  $\kappa = 3.0$  model, with controls computed from Problem (2.18). The  $\kappa = 0.05$  model is used as the source model for transfer learning to  $\kappa = 3.0$  models. Model was trained on  $2.56 \times 10^5$  observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12. The investment setup is detailed in Table 5.1, and the hyper-parameter configuration is provided in Table 5.2.

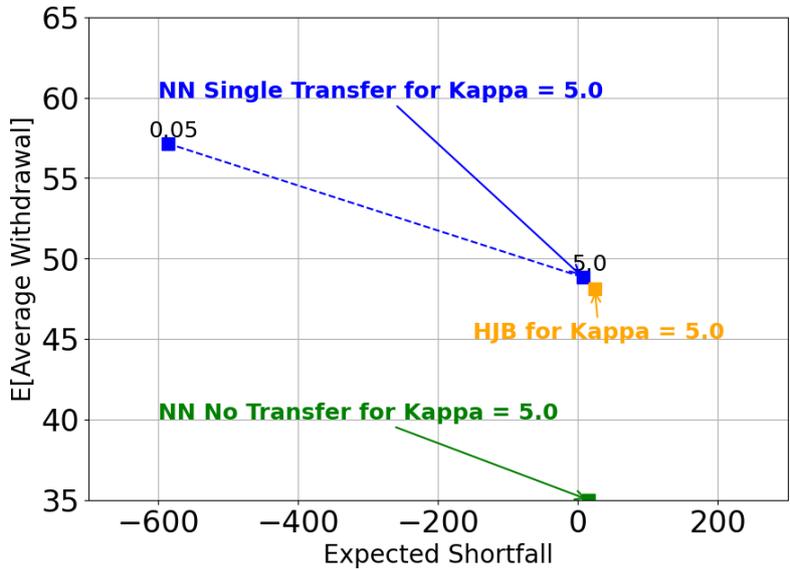


Figure 5.7: Comparison of (EW, ES) outcomes for training  $\kappa = 5.0$  model, with controls computed from Problem (2.18). The  $\kappa = 0.05$  model is used as the source model for transfer learning to  $\kappa = 5.0$  models. Model was trained on  $2.56 \times 10^5$  observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12. The investment setup is detailed in Table 5.1, and the hyper-parameter configuration is provided in Table 5.2.

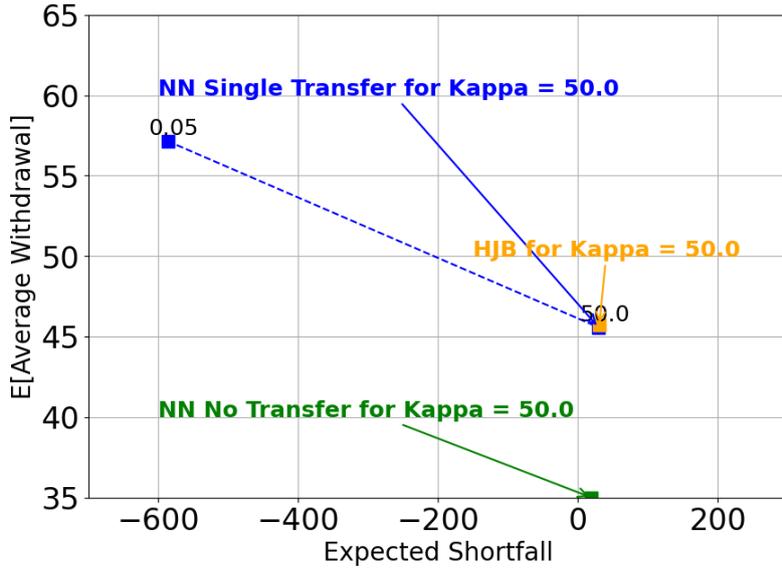


Figure 5.8: Comparison of (EW, ES) outcomes for training  $\kappa = 50.0$  model, with controls computed from Problem (2.18). The  $\kappa = 0.05$  model is used as the source model for transfer learning to  $\kappa = 50.0$  models. Model was trained on  $2.56 \times 10^5$  observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12. The investment setup is detailed in Table 5.1, and the hyper-parameter configuration is provided in Table 5.2.

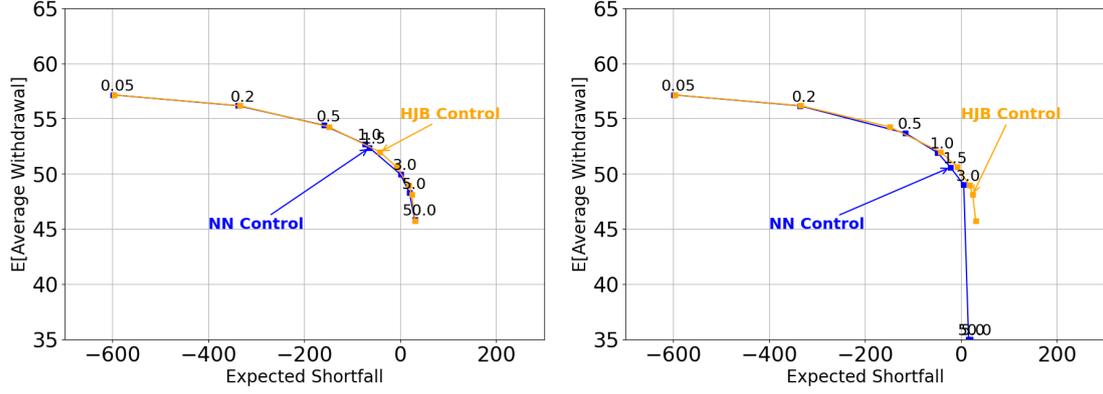
To conclude, the reward-oriented characteristics of the low- $\kappa$  model effectively aid high- $\kappa$  models ( $\kappa = 5.0, 50.0$ ) in mitigating poor training performance caused by the sparsity of tail-end data. However, for models with a more balanced risk-reward preference ( $\kappa = 1.0, 1.5, 3.0$ ), this transfer learning approach incorrectly shifts the training outcome toward a higher reward but higher risk preference.

### Sequential Transfer Learning

To further investigate the impact of transfer learning on NN training, we adopt a sequential transfer learning approach, where each model inherits the trained parameters from the previous  $\kappa$  model rather than transferring directly from  $\kappa = 0.05$ . This iterative approach allows the network to gradually adapt to increasing risk sensitivity, potentially leading to better optimization. The  $\kappa$  values considered in this thesis are [0.05, 0.2, 0.5, 1.0, 1.5, 3.0, 5.0, 50.0].

To evaluate the effectiveness of this approach, we compare the (EW, ES) efficient frontier obtained from sequential transfer learning with the (EW, ES) efficient frontier trained entirely from scratch without transfer learning. This comparison helps determine whether progressive adaptation improves training performance and alignment with the HJB benchmark, particularly for low-risk models  $\kappa = 50$ , which have previously exhibited significant challenges when trained directly.

Figure 5.9 compares the (EW, ES) efficient frontiers obtained using sequential transfer learning (Figure 5.9 (a)) and direct training without transfer learning (Figure 5.9 (b)) against the HJB benchmark. When using sequential transfer learning starting from  $\kappa = 0.05$ , the NN model inherits parameters from previously trained lower  $\kappa$  models, enabling a gradual adaptation to increasing risk sensitivity. As a result, the NN control closely follows the HJB control, with improved alignment across different  $\kappa$  values, especially for  $\kappa = 50.0$ . In contrast, when training without transfer learning, each model is initialized randomly and optimized independently. This leads to significant deviations from the HJB benchmark, particularly at high  $\kappa$  values. The results suggest that progressive transfer learning significantly enhances optimization convergence, particularly for low-risk regions.



(a) EW-ES frontier, transfer sequentially from  $\kappa = 0.05$  model

(b) EW-ES frontier, no transfer

Figure 5.9: Comparison of (EW, ES) efficient frontiers for training performances between sequential transfer learning starting from the  $\kappa = 0.05$  model and no transfer learning by Neural Network (NN) method. Model was trained on  $2.56 \times 10^5$  observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12. Labels on notes indicate  $\kappa$  parameter values for NN results. The investment setup is detailed in Table 5.1, and the hyper-parameter configuration is provided in Table 5.2.

By analyzing the (EW, ES) efficient frontiers in Figure 5.9 (a) and Figure 5.9 (b), we observe that while applying sequential transfer learning starting from  $\kappa = 0.05$  model generally leads to better performance, it also causes certain  $\kappa$  models to deviate from the HJB benchmark.

To further refine this approach, we consider a modified transfer learning approach, where transfer learning is applied starting from an intermediate  $\kappa$  value rather than from the lowest value ( $\kappa = 0.05$ ). Observing the no-transfer (EW, ES) efficient frontier (Figure 5.9 (b)), we find that deviations begin to appear from  $\kappa = 1.5$  onwards. Therefore, we propose starting transfer learning from  $\kappa = 1.0$  model instead, keeping lower  $\kappa$  models independently trained. See results in Table 5.7 and Table 5.8.

$\kappa$	exp 1	exp 2	exp 3	exp 4	exp 5	NN avg obj val	HJB obj val	% difference
0.05	1741.77	1741.76	1741.73	1741.77	1741.77	1741.76	1741.54	0.01%
0.2	1673.79	1673.88	1672.29	1673.60	1673.75	1673.46	1674.41	-0.06%
0.5	1605.75	1605.88	1606.41	1606.51	1605.57	1606.02	1607.26	-0.08%
1.0	1549.58	1567.55	1565.54	1566.09	1566.25	1563.00	1568.45	-0.35%
1.5	1530.20	1555.97	1555.20	1554.24	1554.75	1550.07	1557.46	-0.47%
3.0	1513.34	1564.51	1566.54	1564.56	1565.43	1554.88	1569.71	-0.94%
5.0	1516.89	1604.76	1607.18	1604.99	626.26	1392.02	1612.16	-13.65%
50.0	1973.03	2877.92	2905.65	2895.34	2003.93	2531.17	2946.70	-14.10%

Table 5.7: The objective function values obtained from training with sequential transfer learning starting from  $\kappa = 1.0$  model, with controls computed from Problem (2.18). Model was trained on  $2.56 \times 10^5$  observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12. Investment setup in Table 5.1. Hyper-parameters setup in Table 5.2. Each model is trained five times with different random seeds (0, 1, 2, 3, 4) to examine consistency across runs, refer to Section 5.1.

$\kappa$	ES (5%) (NN)	ES (5%) (HJB)	$E[\sum_i q_i]/(M+1)$ (NN)	$E[\sum_i q_i]/(M+1)$ (HJB)
0.05	-597.91	-596.00	57.15	57.14
0.2	-335.49	-334.29	56.15	56.17
0.5	-122.53	-148.99	53.78	54.25
1.0	-46.71	-42.62	51.93	51.97
1.5	-12.70	-8.05	50.62	50.63
3.0	11.93	17.42	49.01	48.95
5.0	17.87	24.09	48.26	48.12
50.0	25.29	30.60	45.60	45.70

Table 5.8: The averaged expected withdrawal and expected shortfall obtained from training with sequential transfer learning starting from  $\kappa = 1.0$  model, with controls computed from Problem (2.18). Model was trained on  $2.56 \times 10^5$  observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12. Investment setup in Table 5.1. Hyper-parameters setup in Table 5.2. Each model is trained five times with different random seeds (0, 1, 2, 3, 4) to examine consistency across runs, refer to Section 5.1.

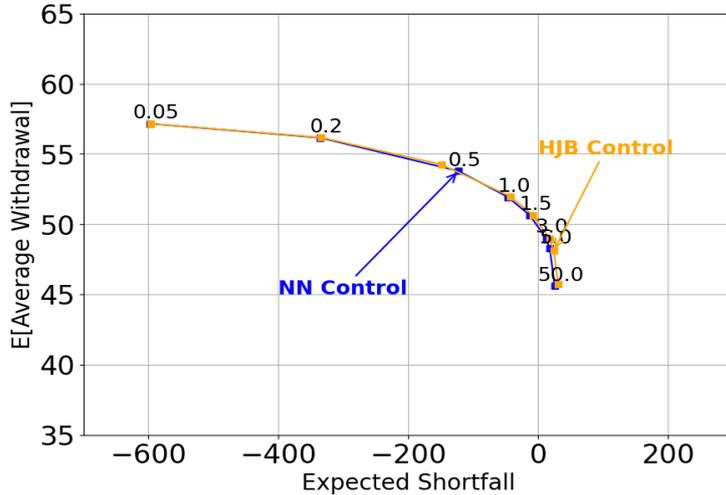


Figure 5.10: (EW, ES) efficient frontier of training performance for sequential transfer learning starting from the  $\kappa = 1.0$  model by Neural Network (NN) method. Labels on notes indicate  $\kappa$  parameter values for NN results. Model was trained on  $2.56 \times 10^5$  observations of simulated data from calibrated jump models for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) from 1926:1 to 2019:12. The investment setup is detailed in Table 5.1, and the hyper-parameter configuration is provided in Table 5.2.

Compared to Figure 5.9 (a), where transfer learning was applied from  $\kappa = 0.05$ , Figure 5.10 demonstrates that initiating sequential transfer learning from  $\kappa = 1.0$  model leads to results that align more closely with the HJB (EW, ES) efficient frontier. This highlights the significant impact of the selection of starting point of transfer learning on the overall (EW, ES) efficient frontier performance.

A crucial observation is that  $\kappa = 1.0$  lies at the *knee* of the efficient frontier, which represents a compromise solution in multi-objective (higher reward and lower risk) optimization problems, balancing expected withdrawals and risk minimization. As a result, using the  $\kappa = 1.0$  model as the foundation for subsequent transfer learning allows the NN to inherit a well-balanced combination of reward-seeking and risk-aware features, leading to better convergence for high- $\kappa$  models. In contrast, we conjecture that the  $\kappa = 0.05$  model initialization primarily focuses on maximizing expected withdrawals, with limited consideration of downside risk.

This finding suggests that carefully selecting the transfer learning starting point is essential in improving the training process for low-risk models. Instead of always transferring from the lowest  $\kappa$ , identifying a well-balanced intermediate point—such as the *knee* of the efficient frontier—allows the model to learn from a more informative and stable optimization trajectory.

### 5.3 Heatmap for Strategy Analysis

We conduct a more detailed analysis by selecting a notable point on the (EW, ES) efficient frontier, corresponding to  $\kappa = 1.0$ . This point is located at the *knee* of the efficient frontier, making it an attractive choice for balancing risk and reward, although the exact selection of  $\kappa$  ultimately depends on investor preference. The concept of the *knee* point is loosely inspired by the compromise solution in multi-objective optimization, which identifies the point on the efficient frontier that minimizes the distance to an ideal yet unattainable solution (Marler and Arora, 2004). For this specific *knee* point at  $\kappa = 1.0$ , we examine the resulting controls and wealth trajectories under both frameworks.

#### 5.3.1 HJB Control Heatmaps

Figure 5.11 illustrates the stock allocation heat maps and wealth percentiles computed using HJB PDE method described in Section 4.2.

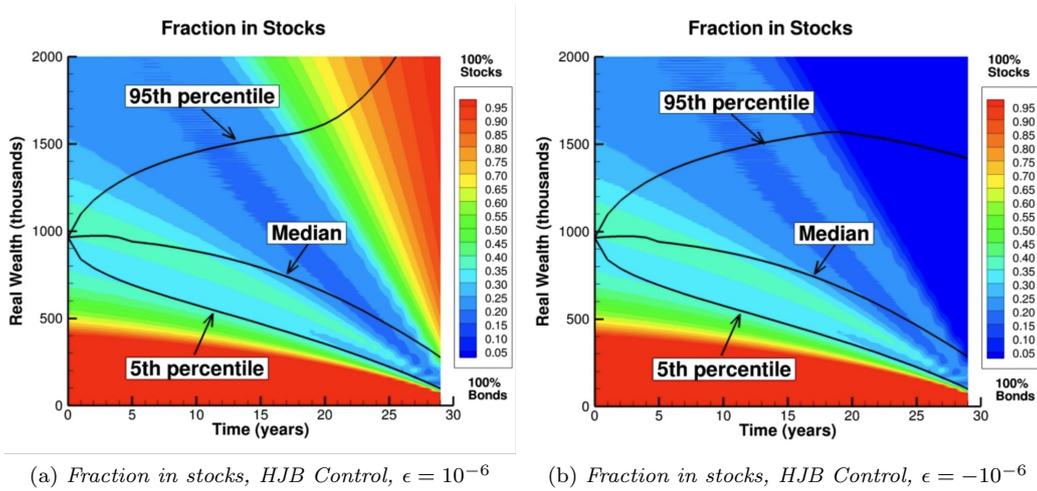


Figure 5.11: *Fraction in stocks heatmap and wealth percentiles with controls computed from Problem (2.18). Note: investment setup is as in Table 5.1. HJB solution performance computed on  $2.56 \times 10^6$  observations of simulated data from calibrated jump models. Parameters for simulated data based on CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD), from 1926:1 to 2019:12, are shown in Table 3.1. Minimum withdrawal: 35. Maximum withdrawal: 60. The wealth percentiles are calculated based on post-withdrawal wealth  $W_t^+$ , while the asset allocation heatmaps are computed from post-withdrawal allocation decisions.  $\kappa = 1.0$ .  $W_0 = 58.0$  for PIDE results. (a)  $\epsilon = 10^{-6}$ . (b)  $\epsilon = -10^{-6}$ . Monetary units: USD\$ in thousands.*

#### 5.3.2 NN Control Heatmaps

For NN control, we have observed three distinctive control heat maps after conducting multiple experiments by using the Hyper-parameters setup in Table 5.2. All the experiments consistently produced expected shortfall and average withdrawal values matching those of the HJB solution, while yielding different control heat maps. Therefore, we believe that the strategies represented by three heat maps are not incorrect results but indicates that there are nearby strategies which generate almost the same (EW, ES) efficient frontiers.

Figure 5.12 illustrates the three control heat maps. The purple area in the bottom-left corner is due to missing data and represents automatic color filling, which can be ignored. Comparing these three results, we can observe that the heat maps remain consistent below the 50th wealth level, with the differences primarily concentrated in the upper portion above the 50th wealth level. By combining the results with wealth percentiles, we can observe that the three different investment strategies have a significant impact on the 95th percentile of wealth, while having minimal influence on the 50th and 25th percentiles. Additionally, in the upper-right portion of the heat map, if we allocate more investment to stocks, the 95th percentile of wealth increases significantly, aligning with the results of the HJB solution.

Since the 5th percentile wealth percentile curves for these three cases remain nearly identical and the withdrawal percentile curves also show minimal differences, their different investment distribution heat maps have little impact on the expected shortfall and average withdrawal. This explains why different investment strategies heat maps give similar average withdrawal, expected shortfall, and objective function value.

Based on above observations, we attribute the NN framework’s inability to fully replicate the HJB control to the ill-posedness of the optimal control problem, due to fact that the objective function is insensitive to the control in the (top-right) region of high wealth levels near  $T$ . The small value of  $\epsilon$  means that the stabilization term contributes a very small fraction of the objective function value and thus has a very small gradient, relative to the first two terms in the objective function (2.18).

To further verify the stability of the control heatmap results for the region except for the top-right region, we increased the absolute value of the stabilization term  $\epsilon W_T$  in the objective function (2.18). Specifically, we adjusted the stabilization parameter  $|\epsilon|$  from  $10^{-6}$  to  $10^{-2}$ . This modification amplifies the influence of expected terminal wealth in the optimization process, ensuring that the model places greater emphasis on stabilizing the final wealth distribution.

Figure 5.13 illustrates the impact of increasing the absolute weight of  $W_T$  in the objective function (2.18) by setting  $\epsilon = 10^{-2}$  and  $\epsilon = -10^{-2}$ . This adjustment amplifies the influence of terminal wealth stabilization, making it a more dominant factor in the optimization process.

Despite this significant modification, the results demonstrate that the lower-left region of the heatmap remains remarkably stable, indicating that the NN control is robust for lower wealth levels and earlier time periods. Specifically, the stock allocation strategy for individuals with lower wealth follows a consistent pattern, suggesting that the learned control maintains a stable response under varying stabilization constraints.

In contrast, the upper-right region exhibits noticeable shifts, reflecting the increased sensitivity of terminal wealth in cases where final wealth levels are extremely high. This suggests that for wealthier individuals nearing the end of the investment horizon, the model’s allocation decisions are more heavily influenced by stabilization constraints, leading to variations in stock allocations.

Overall, these results confirm that even with a significantly increased weight on  $W_T$ , the learned NN control remains stable across most realistic wealth trajectories. The primary variability occurs in high-wealth scenarios, which are inherently more sensitive to the choice of  $\epsilon$ . However, the 95% terminal wealth is not significant in our investment objective.

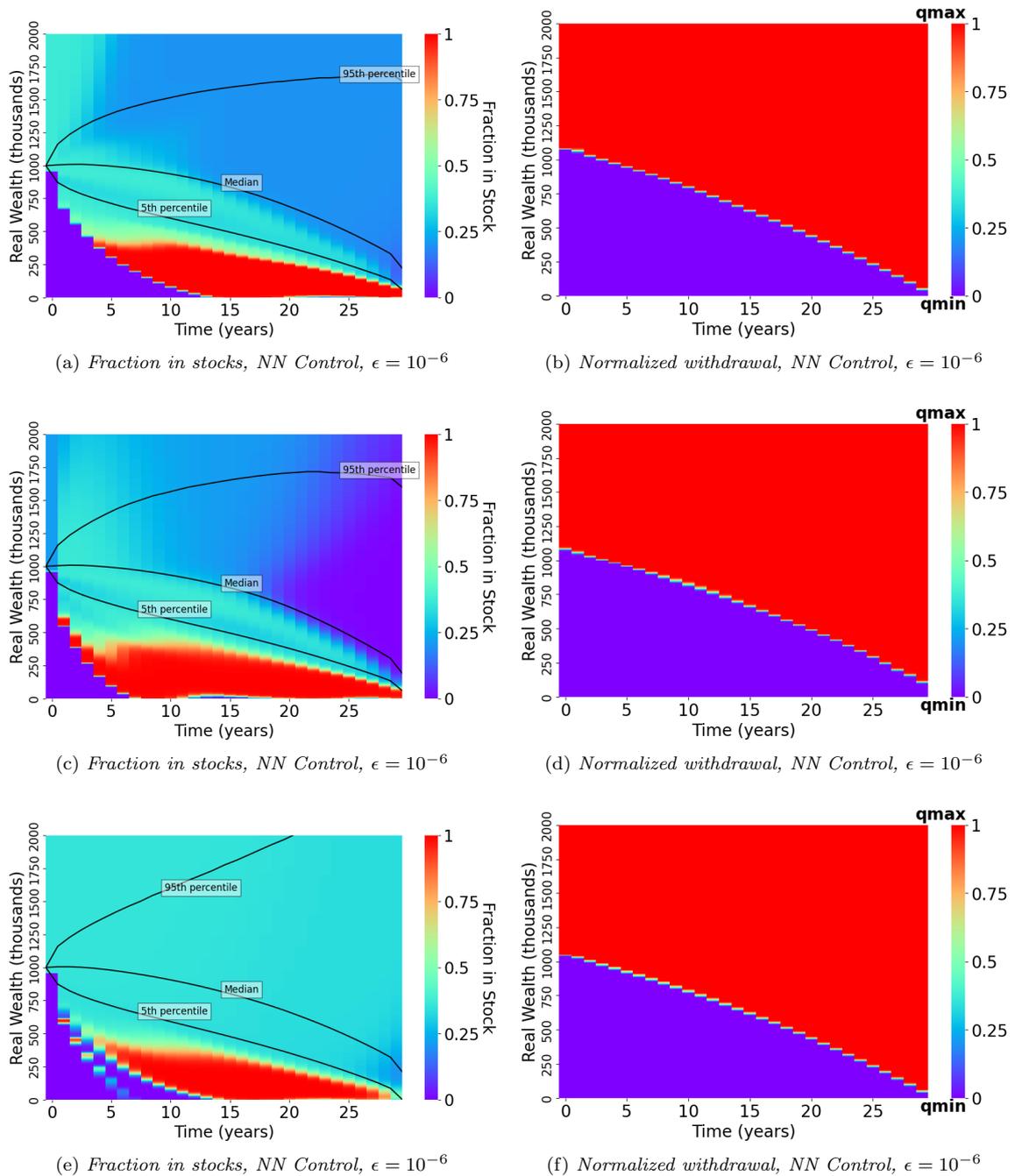


Figure 5.12: Fraction in stocks heatmap and wealth percentiles with controls computed from Problem (2.18). Note: investment setup is as in Table 5.1. Hyper-parameters setup in Table 5.2. NN solution performance trained on  $2.56 \times 10^5$  observations of simulated data from calibrated jump models. Parameters for simulated data based on CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD), from 1926:1 to 2019:12, are shown in Table 3.1. Minimum withdrawal: 35. Maximum withdrawal: 60. The wealth percentiles are calculated based on pre-withdrawal wealth  $W_t^-$ , while the asset allocation heatmaps are computed from post-withdrawal allocation decisions.  $\kappa = 1.0$ . Monetary units: USD\$ in thousands.

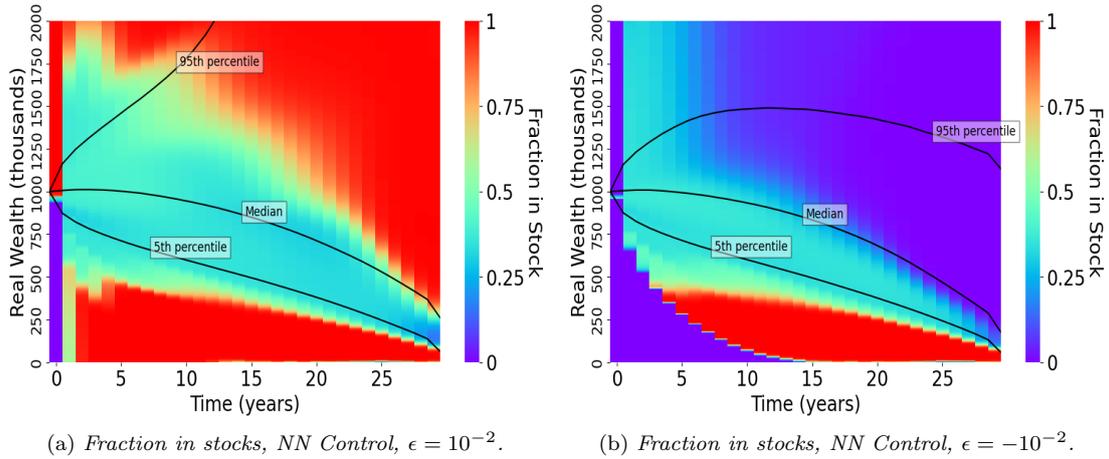


Figure 5.13: *Fraction in stocks heatmap and wealth percentiles with controls computed from Problem (2.18). Note: investment setup is as in Table 5.1. Hyper-parameters setup in Table 5.2. NN solution performance trained on  $2.56 \times 10^5$  observations of simulated data from calibrated jump models. Parameters for simulated data based on CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD), from 1926:1 to 2019:12, are shown in Table 3.1. Minimum withdrawal: 35. Maximum withdrawal: 60. The wealth percentiles are calculated based on pre-withdrawal wealth  $W_t^-$ , while the asset allocation heatmaps are computed from post-withdrawal allocation decisions.  $\kappa = 1.0$ . Monetary units: USD\$ in thousands.*

## 5.4 Updated Historical Data Training Results

We conduct asset return and NN training using the updated CRSP dataset spanning from January 1926 to December 2023. We compare the NN performance on this extended dataset with results from the original CRSP data (1926:1–2019:12) and analyze the asset returns of these two datasets in this section.

### 5.4.1 Asset Return

Return is a fundamental metric used to measure the price movement of assets. It plays a crucial role in financial analysis, investment decisions, and risk management. The two most common methods for calculating returns are *Simple Return* and *Log Return*. While these methods often yield similar results for small price changes, they exhibit significant differences in mathematical properties and application scenarios.

#### 1. Simple Return

- (a) **Definition:** Simple return represents the relative percentage change in asset price over a given period.
- (b) **Formula:** The simple return for period  $t$  is given by:

$$R_t = \frac{P_t - P_{t-1}}{P_{t-1}} \quad (5.3)$$

where:

- $R_t$  is the simple return at time  $t$ ;
  - $P_t$  is the asset price at time  $t$ ;
  - $P_{t-1}$  is the asset price at time  $t - 1$ .
- (c) **Multi-Period Computation:** For multiple periods, simple returns are compounded multiplicatively:

$$\text{Total Return} = (1 + R_1) \times (1 + R_2) \times \cdots \times (1 + R_n) - 1 \quad (5.4)$$

## 2. Log Return

- (a) **Definition:** Log return is the natural logarithm of the ratio of an asset’s price between two periods.
- (b) **Formula:** The log return for period  $t$  is given by:

$$r_t = \ln\left(\frac{P_t}{P_{t-1}}\right) \quad (5.5)$$

where:

- $r_t$  is the log return at time  $t$ ;
  - $\ln(\cdot)$  represents the natural logarithm function.
- (c) **Multi-Period Computation:** Log returns can be summed directly over multiple periods:

$$R_{\text{total}} = r_1 + r_2 + \cdots + r_n \quad (5.6)$$

The total return can be obtained by exponentiating the sum:

$$\text{Total Return} = e^{R_{\text{total}}} - 1 \quad (5.7)$$

### 5.4.2 Computational Results

To further investigate the performance of the neural network framework on the updated CRSP historical dataset, we employ a bootstrap resampling approach described in Section 3.4, with expected block size of 3 (see Table 3.2), to generate training samples from both the old dataset (January 1926 to December 2019) and the new dataset (January 1926 to December 2023). Using the hyper-parameter settings specified in Table 5.2, we then train the neural network separately on both *resampled datasets*. This analysis aims to assess the NN framework’s adaptability to newly incorporated data and examine any deviations in performance between the two training periods. Table 5.9, 5.10, and 5.11 show the performance of NN training on bootstrap resampling new historical data set from 1926:1 to 2023:12.

$\kappa$	exp 1	exp 2	exp 3	exp 4	exp 5	NN avg obj val
0.05	1737.57	1737.60	1737.55	1737.56	1737.10	1737.48
0.2	1668.36	1668.48	1668.49	1668.24	1667.79	1668.27
0.5	1589.47	1588.89	1589.55	1570.75	1587.89	1585.31
1.0	1528.18	1525.59	1506.05	1526.24	1526.28	1522.47
1.5	1482.70	1480.88	1454.13	1488.89	1478.62	1477.04
3.0	1361.83	1361.02	1313.07	1416.74	1356.61	1361.85
5.0	244.32	1345.99	1245.33	1348.16	1338.35	1104.43
50.0	173.72	43.92	-847.41	188.21	103.09	-67.69

Table 5.9: *Objective function values for NN training performance on CPI adjusted CRSP 10-Year T-Bond Returns (B10) & CRSP Value-Weighted Index (VWD), from 1926:1 to 2023:12, with controls computed from Problem (2.18). Model trained on  $2.56 \times 10^5$  observations of resampled data from bootstrap resampling. Investment setup in Table 5.1. Hyper-parameters setup in Table 5.2. Expected block size of bootstrap resampling is 3, see Table 3.2. Each model is trained five times with different random seeds (0, 1, 2, 3, 4) to examine consistency across runs, refer to Section 5.1.*

$\kappa$	exp 1	exp 2	exp 3	exp 4	exp 5	NN avg expected shortfall
0.05	-586.65	-587.05	-586.04	-585.56	-588.03	-586.67
0.2	-373.64	-371.66	-372.19	-372.20	-373.80	-372.70
0.5	-201.72	-183.82	-201.15	-152.74	-186.08	-185.10
1.0	-97.87	-97.75	-111.24	-93.84	-99.48	-100.04
1.5	-86.56	-85.97	-100.87	-62.82	-86.74	-84.59
3.0	-75.73	-75.58	-90.91	-38.95	-75.00	-71.23
5.0	-72.79	-32.05	-52.87	-31.64	-33.39	-44.55
50.0	-25.08	-27.52	-45.64	-24.80	-26.56	-29.92

Table 5.10: *Expected shortfall for NN training performance on CPI adjusted CRSP 10-Year T-Bond Returns (B10) & CRSP Value-Weighted Index (VWD), from 1926:1 to 2023:12, with controls computed from Problem (2.18). Model trained on  $2.56 \times 10^5$  observations of resampled data from bootstrap resampling. Investment setup in Table 5.1. Hyper-parameters setup in Table 5.2. Expected block size of bootstrap resampling is 3, see Table 3.2. Each model is trained five times with different random seeds (0, 1, 2, 3, 4) to examine consistency across runs, refer to Section 5.1.*

$\kappa$	exp 1	exp 2	exp 3	exp 4	exp 5	NN avg expected withdrawal
0.05	57.00	57.00	57.00	56.99	56.98	56.99
0.2	56.23	56.22	56.22	56.22	56.21	56.22
0.5	54.53	54.22	54.52	53.63	54.22	54.22
1.0	52.45	52.37	52.17	52.26	52.44	52.34
1.5	52.02	51.93	51.79	51.07	51.89	51.74
3.0	51.26	51.22	51.15	49.47	51.02	50.82
5.0	50.80	48.59	48.70	48.59	48.56	49.05
50.0	46.08	46.56	46.27	46.07	46.17	46.23

Table 5.11: *Expected withdrawal for NN training performance on CPI adjusted CRSP 10-Year T-Bond Returns (B10) & CRSP Value-Weighted Index (VWD), from 1926:1 to 2023:12, with controls computed from Problem (2.18). Model trained on  $2.56 \times 10^5$  observations of resampled data from bootstrap resampling. Investment setup in Table 5.1. Hyper-parameters setup in Table 5.2. Expected block size of bootstrap resampling is 3, see Table 3.2. Each model is trained five times with different random seeds (0, 1, 2, 3, 4) to examine consistency across runs, refer to Section 5.1.*

Figure 5.14 shows that when incorporating data from 2020 to 2023 (blue), the NN control exhibits significant deviations, particularly in the low-risk region (right-end of the frontier). The shape of the (EW, ES) efficient frontier shifts downward, indicating that the model suggests more conservative withdrawal strategies under higher risk sensitive levels. This divergence suggests that the recent market dynamics have significantly influenced the computational results. However, in the high-risk region (left-end of the frontier), the two frontiers remain close, implying that for higher reward investment strategies, the additional years of data do not substantially impact the withdrawal policy.

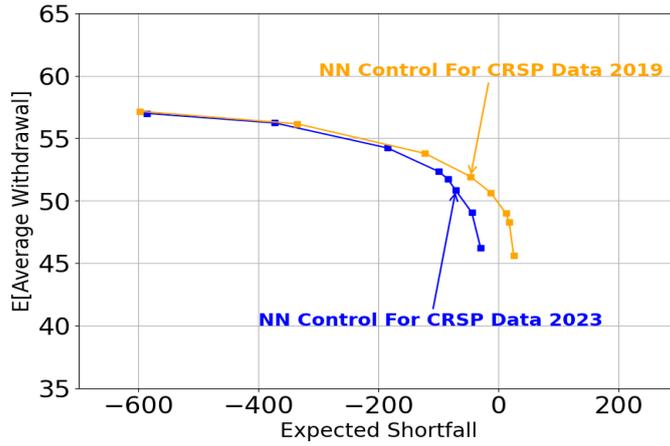
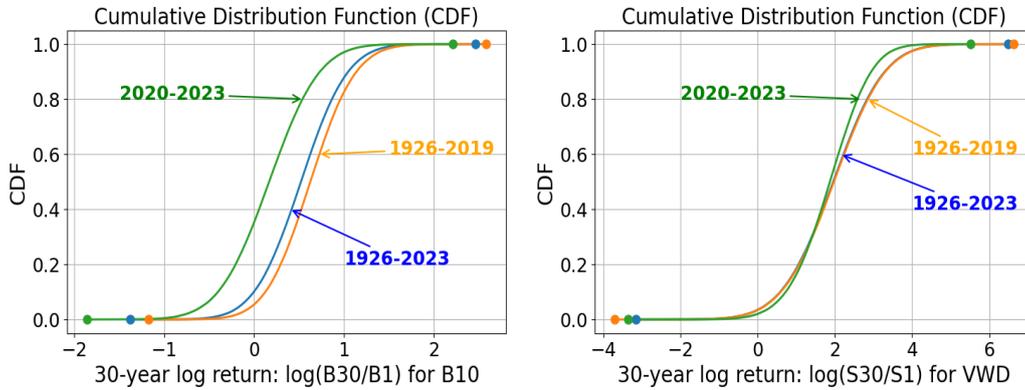


Figure 5.14: Comparison of (EW, ES) efficient frontiers for NN training performance on CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) over two different time spans: 1926:1 - 2019:12 and 1926:1 - 2023:12. Controls are computed from Problem (2.18). Model trained on  $2.56 \times 10^5$  observations of resampled data from bootstrap resampling. Investment setup is detailed in Table 5.1, and hyper-parameter configurations are provided in Table 5.2. Expected block size of bootstrap resampling is 3, see Table 3.2.

To explore the factors contributing to the differences between using resampled data from CRSP 1926:1–2019:12 and CRSP 1926:1–2023:12, we analyze the cumulative distribution function (CDF) of 30-year log returns for both resampled datasets.



(a) CRSP 10-Year T-Bond Returns (B10) 30-year log-return cdf plot (b) CRSP Value-Weighted Index (VWD) 30-year log-return cdf plot

Figure 5.15: Comparison of the cumulative distribution functions (CDF) of 30-year log returns for CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) across different time spans. The datasets are generated using  $2.56 \times 10^5$  observations of bootstrap resampling data from CRSP 1926:1–2019:12, CRSP 1926:1–2023:12, and CRSP 2020:1–2023:12. Expected block size of bootstrap resampling is 3, see Table 3.2.

Figure 5.15 illustrates the impact of different data periods on the distribution of 30-year log returns for CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD). The datasets used in this analysis are generated using Bootstrap Resampling from CPI adjusted CRSP 1926:1–2019:12 historical data and CRSP 1926:1–2023:12 historical data. In the B10 CDF plot (Figure 5.15 (a)), we can see that the proportion of returns with  $\log(B30/B1) \leq 0.5$  is approximately 40% for the 1926:1-2019:12 period (orange curve); the proportion of returns with  $\log(B30/B1) \leq 0.5$  is approximately 50% for the 1926:1-2023:12 period (blue curve); the proportion of returns with  $\log(B30/B1) \leq 0.5$  is approximately 80% for the 2020:1-2023:12 period (green curve).

curve). It indicates that the inclusion of the most recent four years (2020:1-2023:12) significantly affects the overall distribution. In the VWD CDF plot (Figure 5.15 (b)), we can see that the proportion of returns with  $\log(S_{30}/S_1) \leq 3.0$  is approximately 80% for the 1926:1-2019:12 period (orange curve); the proportion of returns with  $\log(S_{30}/S_1) \leq 3.0$  is also approximately 80% for the the 1926:1-2023:12 period (blue curve); the proportion of returns with  $\log(S_{30}/S_1) \leq 3.0$  is approximately 90% for the 2020:1-2023:12 period (green curve).

The shift in Figure 5.15 (a) for B10 can be explained by the dramatic change in financial conditions during the 2020–2023 period. From 1985 to 2020, the 10-year bond interest rate exhibited a long-term stable downward trend, which supported rising bond prices and, consequently, relatively higher long-term returns. However, starting in 2020, the interest rate environment changed abruptly. Following the initial rate cuts during the onset of the COVID-19 pandemic, inflation surged, prompting the Federal Reserve to implement aggressive rate hikes from 2022 onward. As a result, the 10-year bond interest rate increased sharply, leading to a significant decline in bond prices. This substantial drop in bond prices over the 2020–2023 period caused the bootstrap resampled CRSP 30-year log returns for B10 to decline sharply as well, thereby shifting the cumulative distribution function (CDF) of the bootstrap resampled CRSP 30-year log returns for B10 over 2020 - 2023 to the left.

To assess the impact of asset returns conducted in Figure 5.15 on the deviation observed in the right-end of the (EW, ES) efficient frontier in Figure 5.14, we analyze the allocation percentiles of CRSP 10-Year T-Bond Returns (B10) for both the  $\kappa = 0.05$  and  $\kappa = 50.0$  models, as shown in Figure 5.16.

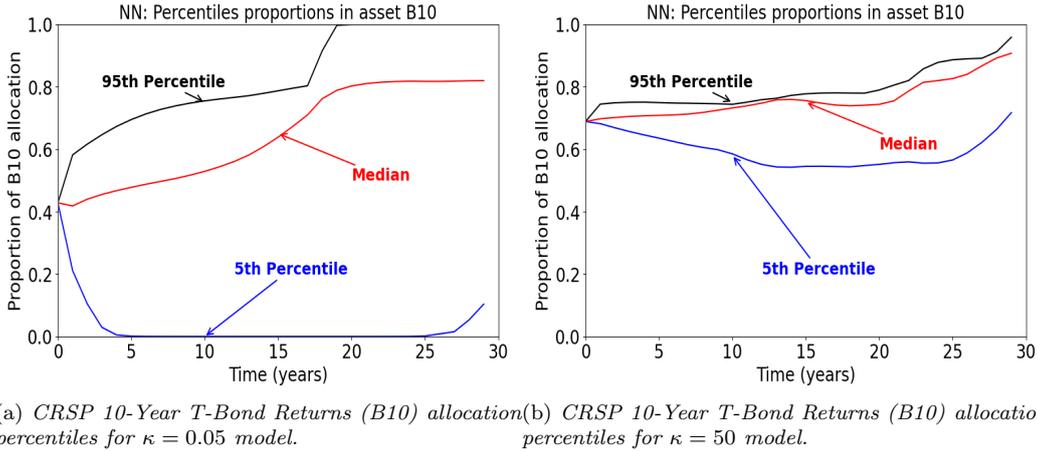


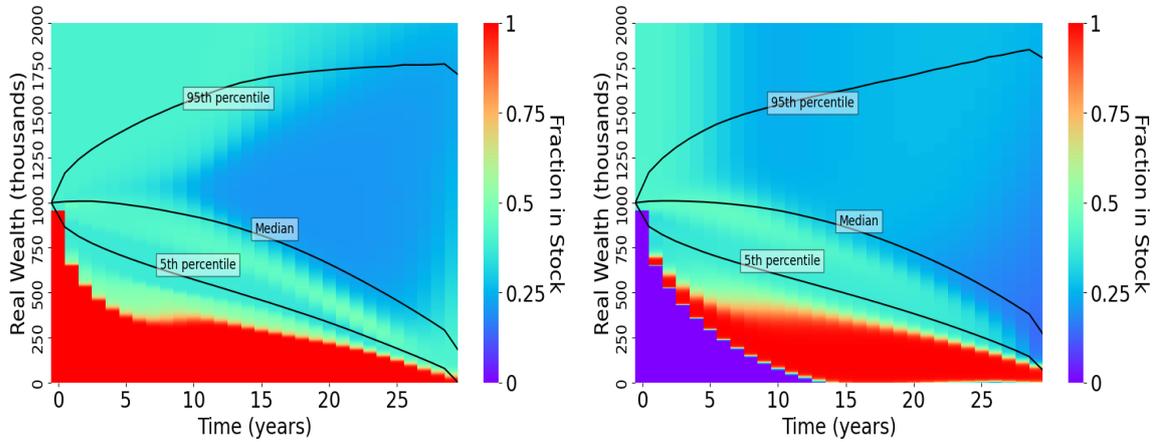
Figure 5.16: B10 allocation percentiles for NN training performance on CPI adjusted CRSP 10-Year T-Bond Returns (B10) and CRSP Value-Weighted Index (VWD) over 1926:1 - 2023:12. Controls are computed from Problem (2.18). Model trained on  $2.56 \times 10^5$  observations of resampled data from bootstrap resampling. Investment setup is detailed in Table 5.1, and hyper-parameter configurations are provided in Table 5.2. Expected block size of bootstrap resampling is 3, see Table 3.2.

Figure 5.16 illustrates that as  $\kappa$  increases, the allocation proportion in CRSP 10-Year T-Bond Returns (B10) significantly rises. Figure 5.16 (a) shows the percentile allocation trends for a high-risk regions (small  $\kappa$  models, e.g.  $\kappa = 0.05$ ), where the 5th percentile (blue) rapidly drops close to zero in the early years, while the median (red) and 95th percentile (black) allocations gradually increase. In contrast, for the low-risk regions (large  $\kappa$  models, e.g.  $\kappa = 50.0$ ), Figure 5.16 (b) indicates that the allocation to B10 remains consistently high across all percentiles, with the 5th percentile maintaining a much higher allocation than in the low- $\kappa$  model.

This increased allocation in B10 under higher  $\kappa$  values makes the model more sensitive to the distributional shifts in the bond market (B10) shown in Figure 5.15 (a). This shift in B10 returns directly affects the (EW, ES) efficient frontier performance of the NN model, as seen in Figure 5.14. For high-risk regions (low  $\kappa$ ), the two efficient frontiers remain relatively close.

However, at low-risk regions (higher  $\kappa$ ), where B10 plays a dominant role in asset allocation, the two efficient frontiers diverge sharply. This suggests that the increased weight on B10 amplifies the impact of asset return difference and explains how the asset returns influence the (EW, ES) frontier performance.

Figure 5.17 compares the stock allocation heatmaps and wealth percentiles for the NN control using Bootstrap Resampling CRSP historical data between 1926:1 - 2019:12 and 1926:1 - 2023:12, respectively. Despite some differences in the return distributions of B10 between these two datasets (as shown in Figure 5.15 (a)), the stock allocation heatmap patterns remain highly consistent: The lower-left region of both heat maps, representing lower wealth percentiles, exhibits nearly identical allocation fraction in stock (except for the automatic color filling in the bottom-left corner). Similarly, in the upper-right region, corresponding to higher wealth levels, the proportion allocated to stocks remains stable across both datasets. This consistency indicates that although the most recent four years of data may have influenced the estimated return distributions (Figure 5.15 (a)), they have not significantly altered the optimal investment strategy for the  $\kappa = 1.0$  model.



(a) Fraction in stocks, NN Control, CRSP data up to 2023. (b) Fraction in stocks, NN Control, CRSP data up to 2019.

Figure 5.17: *Fraction in stocks heatmap and wealth percentiles with controls computed from Problem (2.18). Note: investment setup is as in Table 5.1. Hyper-parameters setup in Table 5.2. NN solution performance computed on  $2.56 \times 10^5$  observations of bootstrap resampling CPI adjusted CRSP historical data (CRSP 10-Year T-Bond Returns (B10) & CRSP Value-Weighted Index (VWD)) from 1926:1 to 2023:12. The wealth percentiles are calculated based on pre-withdrawal wealth  $W_t^-$ , while the asset allocation heatmaps are computed from post-withdrawal allocation decisions. Expected block size of bootstrap resampling is 3, see Table 3.2.  $\kappa = 1.0$ . Monetary units: USD\$ in thousands.*

## 5.5 Multi-Assets Training Results

We extend our analysis by incorporating other CRSP assets ((i) CRSP 30-Day T-Bill Returns (T30), (ii) CRSP 90-Day T-Bill Returns (T90), (iii) CRSP Equal-Weighted Index(VWD)), exploring a wider range of historical financial data. This allows for a more comprehensive comparison of the characteristics of different assets and their impact on the NN training results. By examining variations in asset return distributions, we aim to identify how these differences influence the learned investment strategies. The results will provide further insights into the High-Dimensional Applicability of NN framework under multi-assets environments.

**Bond Assets:** CRSP 10-Year T-Bond Returns (B10), CRSP 30-Day T-Bill Returns (T30), and CRSP 90-Day T-Bill Returns (T90) are analyzed first. This analysis focuses on their return distributions and each bond type is conducted comparative training analyses with the CRSP

Value-Weighted Index (VWD) using the NN control. This helps assess how different bond choices influence the overall investment strategy and risk-reward trade-offs when combined with equity investments.

Figure 5.18 illustrates the distribution of 30-year log returns for CRSP 30-Day T-Bill Returns (T30), CRSP 90-Day T-Bill Returns (T90), and CRSP 10-Year T-Bond Returns (B10). The datasets used in this analysis are generated using Bootstrap Resampling from CPI adjusted CRSP 1926:1–2023:12 historical data. We can see that the proportion of returns with  $\log(B30/B1) \leq 0.5$  is approximately 50% for B10 asset (blue curve); the proportion of returns with  $\log(B30/B1) \leq 0.5$  is close to 100% for T90 asset (green curve); the proportion of returns with  $\log(B30/B1) \leq 0.5$  is approximately 100% for T90 asset (orange curve). It demonstrates that longer-term bonds (B10) exhibit greater return potential over extended periods, while shorter-term bonds (T30, T90) are almost entirely constrained to lower return levels.

Figure 5.19 illustrates the (EW, ES) efficient frontiers for neural network (NN) controls trained with different bond assets—CRSP 10-Year T-Bond Returns (B10), CRSP 30-Day T-Bill Returns (T30), and CRSP 90-Day T-Bill Returns (T90)—each combined with the CRSP Value-Weighted Index (VWD). The results indicate that, for a given expected shortfall, the B10 model (blue curve) consistently achieves the highest expected withdrawal, particularly in regions where risk and reward are balanced. This observation aligns with the cumulative distribution function (CDF) analysis in Figure 5.18, which highlights B10’s superior long-term return potential. As risk aversion increases, the differences among bond selections become more pronounced, underscoring the significant role of bond choice under risk-sensitive conditions. These findings confirm that long-term bonds (B10) offer more favorable investment opportunities than short-term bonds (T30 and T90).

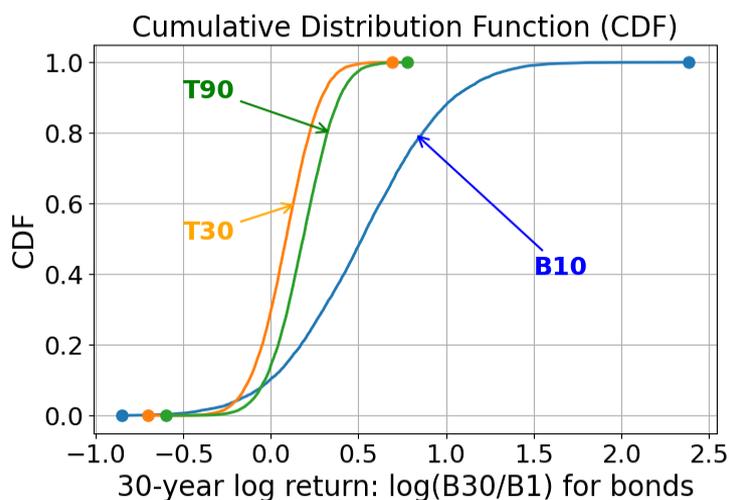


Figure 5.18: Comparison of the cumulative distribution functions (CDF) of 30-year log returns for CPI adjusted CRSP 30-Day T-Bill Returns (T30), CRSP 90-Day T-Bill Returns (T90), and CRSP 10-Year T-Bond Returns (B10), from 1926:1 to 2023:12. The datasets are generated using  $2.56 \times 10^5$  observations of bootstrap resampled data. Expected block size of bootstrap resampling is 3, see Table 3.2.

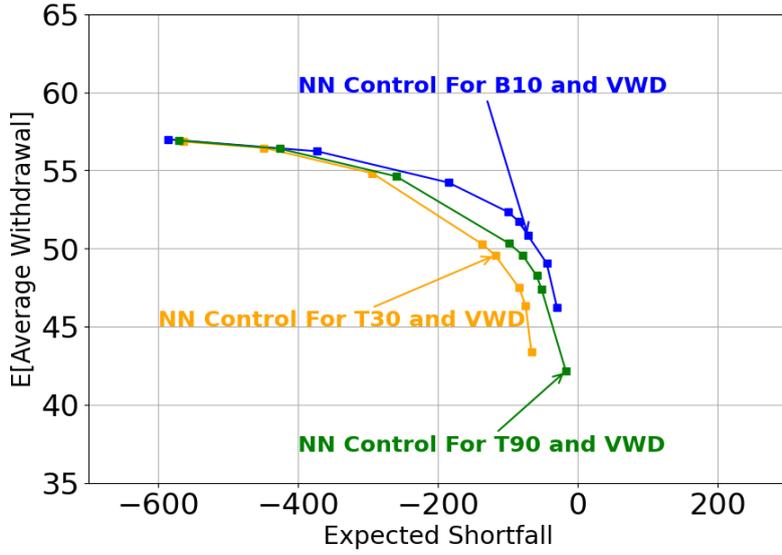
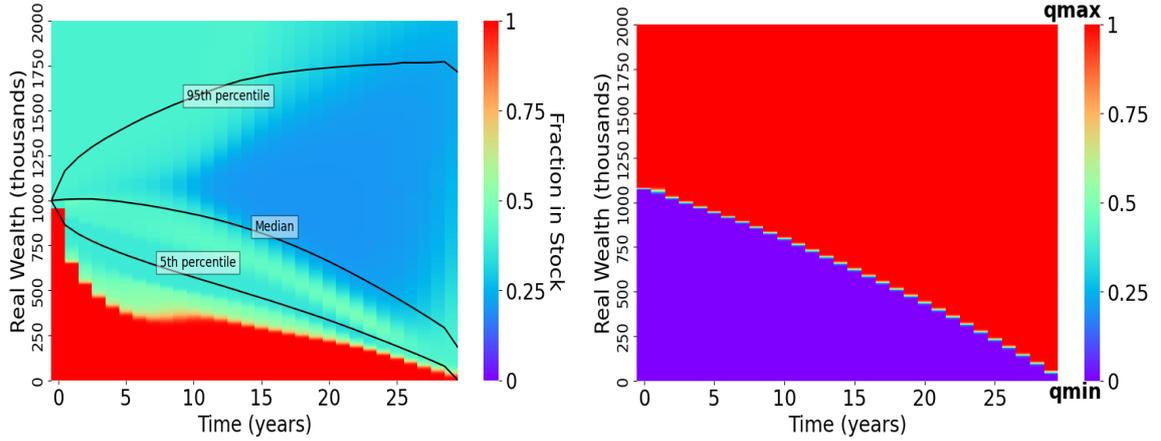
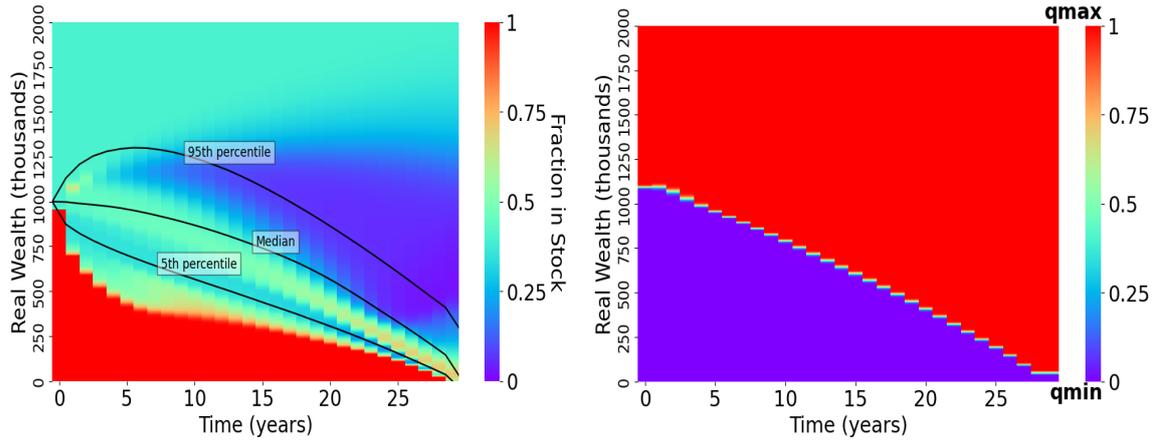


Figure 5.19: Comparison of (EW, ES) efficient frontiers for the Neural Network (NN) training for different CPI adjusted CRSP bond assets : CRSP 10-Year T-Bond Returns (B10) & CRSP 30-Day T-Bill Returns (T30) & CRSP 90-Day T-Bill Returns (T90), together with CPI adjusted CRSP Value-Weighted Index (VWD), computed from the Problem (2.18). Note: investment setup in Table 5.1. Hyper-parameters setup in Table 5.2. Control computed from the NN model, trained on  $2.56 \times 10^5$  observations of bootstrapping resampling CRSP historical data for each bond asset from 1926:1 to 2023:12. Expected block size of bootstrap resampling is 3, see Table 3.2.



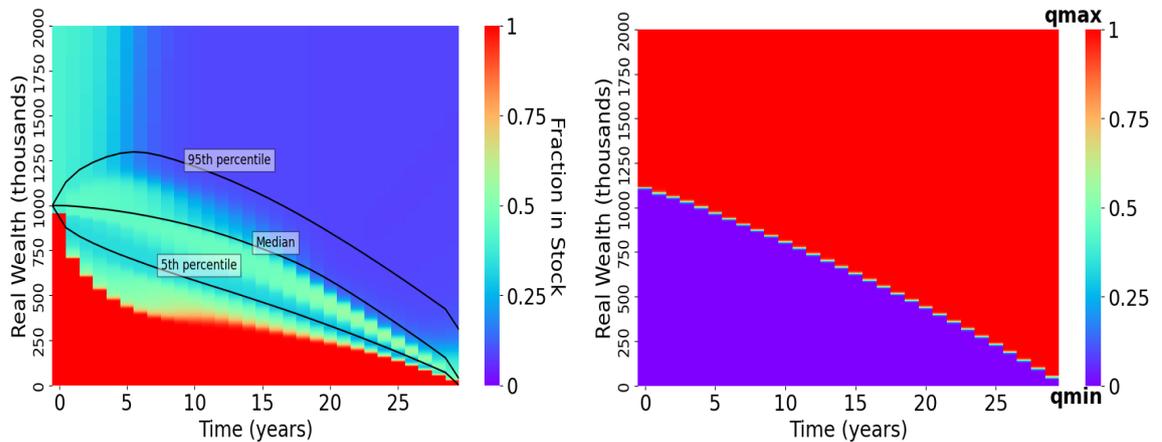
(a) Fraction in stocks, NN Control, CRSP data up to 2023. (b) Normalized withdrawal, NN Control, CRSP data up to 2023.

Figure 5.20: Fraction in stocks heatmap and wealth percentiles, and normalized withdrawal with controls computed from Problem (2.18). Note: investment setup is as in Table 5.1. Hyper-parameters setup in Table 5.2. NN solution performance computed on  $2.56 \times 10^5$  observations of bootstrap resampling CPI adjusted CRSP historical data (CRSP 10-Year T-Bond Returns (B10) & CRSP Value-Weighted Index (VWD)) from 1926:1 to 2023:12. The wealth percentiles are calculated based on pre-withdrawal wealth  $W_t^-$ , while the asset allocation heatmaps are computed from post-withdrawal allocation decisions. Expected block size of bootstrap resampling is 3, see Table 3.2.  $\kappa = 1.0$ . Monetary units: USD\$ in thousands.



(a) Fraction in stocks, NN Control, CRSP data up to 2023. (b) Normalized withdrawal, NN Control, CRSP data up to 2023.

Figure 5.21: Fraction in stocks heatmap and wealth percentiles, and normalized withdrawal, with controls computed from Problem (2.18). Note: investment setup is as in Table 5.1. Hyperparameters setup in Table 5.2. NN solution performance computed on  $2.56 \times 10^5$  observations of bootstrap resampling CPI adjusted CRSP historical data (CRSP 30-Day T-Bill Returns (T30) & CRSP Value-Weighted Index (VWD)) from 1926:1 to 2023:12. The wealth percentiles are calculated based on pre-withdrawal wealth  $W_t^-$ , while the asset allocation heatmaps are computed from post-withdrawal allocation decisions. Expected block size of bootstrap resampling is 3, see Table 3.2. Expected block size of bootstrap resampling is 3, see Table 3.2.  $\kappa = 1.0$ . Monetary units: USD\$ in thousands.



(a) Fraction in stocks, NN Control, CRSP data up to 2023. (b) Normalized withdrawal, NN Control, CRSP data up to 2023.

Figure 5.22: Fraction in stocks heatmap and wealth percentiles, and normalized withdrawal, with controls computed from Problem (2.18). Note: investment setup is as in Table 5.1. Hyperparameters setup in Table 5.2. NN solution performance computed on  $2.56 \times 10^5$  observations of bootstrap resampling CPI adjusted CRSP historical data (CRSP 90-Day T-Bill Returns (T90) & CRSP Value-Weighted Index (VWD)) from 1926:1 to 2023:12. The wealth percentiles are calculated based on pre-withdrawal wealth  $W_t^-$ , while the asset allocation heatmaps are computed from post-withdrawal allocation decisions. Expected block size of bootstrap resampling is 3, see Table 3.2.  $\kappa = 1.0$ . Monetary units: USD\$ in thousands.

The three figures (Figure 5.20, 5.21, 5.22) illustrate the fraction of stocks allocation heatmaps and the corresponding normalized withdrawal patterns for investment strategies incorporating different CRSP bonds: CRSP 10-Year T-Bond Returns (B10), CRSP 30-Day T-Bill Returns (T30), and CRSP 90-Day T-Bill Returns (T90). Despite the differences in the return distributions of these three bond types in Figure 5.18, their investment heatmaps exhibit remarkably similar patterns. This consistency in allocation patterns indicates that the neural network (NN) control follows similar investment strategies across all three bond types, reinforcing the robustness of the optimal control problem.

A key observation is that in the upper-right region of the heatmaps—corresponding to near terminal high wealth scenarios—the fraction of stock investments remains similar across all three bond types. This suggests that, for high-wealth scenarios, the allocation strategy is less sensitive to the specific bond type chosen.

However, a significant divergence emerges when examining the 95th percentile wealth trajectory. Due to the relatively higher returns associated with B10, its upper quantile wealth trajectory is visibly elevated compared to T30 and T90. This outcome is consistent with the findings from the cumulative distribution function (CDF) analysis in 5.18, where B10 exhibited a higher probability of achieving large long-term returns relative to T30 and T90. The superior performance of B10 in terms of wealth accumulation underscores the impact of higher bond returns on long-term investment outcomes, even when the overall asset allocation strategy remains structurally similar across different bond choices.

**Equity assets:** CRSP Value-Weighted Index (VWD), and CRSP Equal-Weighted Index (EWD) are analyzed next. This analysis focuses on their return distributions and each stock type is conducted comparative analyses using NN controls trained with the CRSP 10-Year T-Bond Returns (B10) and chosen equity index. This helps assess how different equity index choices influence the overall investment strategy and risk-reward trade-offs when combined with bond investments.

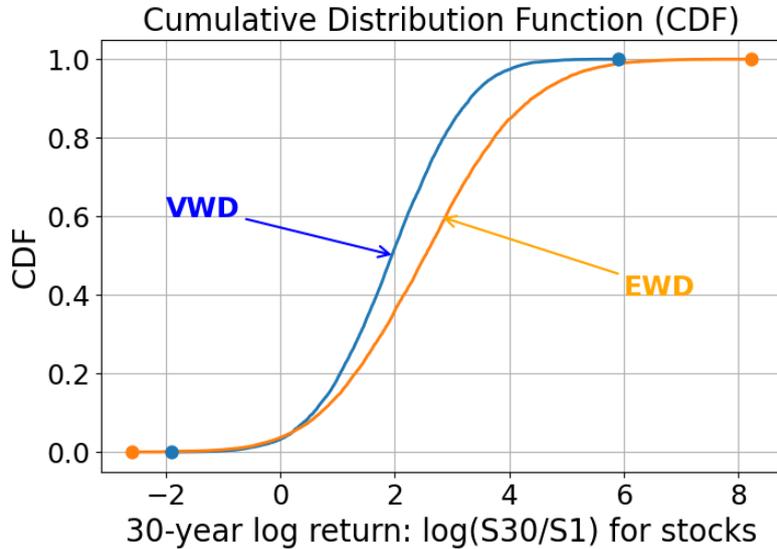


Figure 5.23: Comparison of the cumulative distribution functions (CDF) of 30-year log returns for CPI adjusted CRSP Value-Weighted Index (VWD), and CRSP Equal-Weighted Index (EWD), from 1926:1 to 2023:12. The datasets are generated from  $2.56 \times 10^5$  bootstrap resampled data. Expected block size of bootstrap resampling is 3, see Table 3.2.

Figure 5.23 illustrates the distribution of 30-year log returns for CRSP Value-Weighted Index (VWD), and CRSP Equal-Weighted Index (EWD). The datasets used in this analysis are generated using Bootstrap Resampling from CPI adjusted CRSP 1926:1–2023:12 historical data. We can see that the proportion of returns with  $\log(S_{30}/S_1) \leq 3.0$  is approximately 80% for VWD asset (blue

curve); the proportion of returns with  $\log(S_{30}/S_1) \leq 3.0$  is approximately 60% for EWD asset (green curve). It demonstrates that EWD asset tends to have higher long-term return potential compared to VWD asset.

Figure 5.24 illustrates the (EW, ES) efficient frontiers for neural network (NN) controls trained with different equity assets—CRSP Value-Weighted Index (VWD), and CRSP Equal-Weighted Index (EWD)—each combined with the CRSP 10-Year T-Bond Returns (B10).

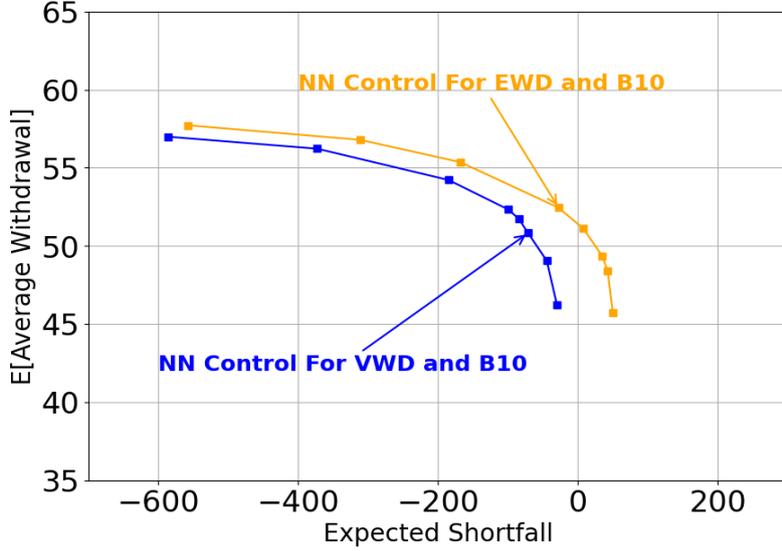
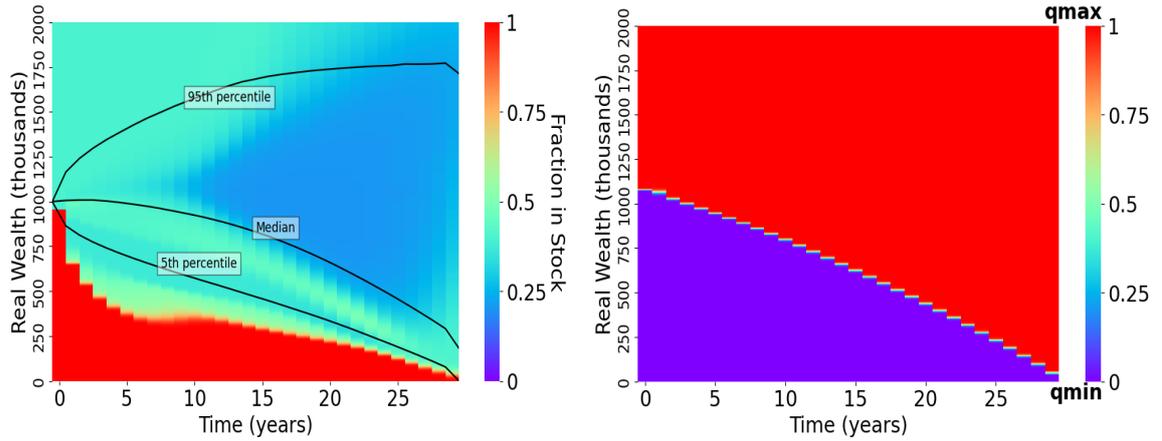


Figure 5.24: Comparison of (EW, ES) efficient frontiers for the Neural Network (NN) training for different CPI adjusted CRSP equity assets : CRSP Value-Weighted Index (VWD) & CRSP Equal-Weighted Index (EWD), together with CPI adjusted CRSP 10-Year T-Bond Returns (B10), computed from the Problem (2.18). Note: investment setup in Table 5.1. Hyper-parameters setup in Table 5.2. Control computed from the NN model, trained on  $2.56 \times 10^5$  observations of bootstrapping resampling CRSP historical data for each equity asset from 1926:1 to 2023:12. Expected block size of bootstrap resampling is 3, see Table 3.2.

Figures 5.23 and 5.24 illustrate significant differences between using CRSP Value-Weighted Index (VWD) and CRSP Equal-Weighted Index (EWD) in both the cumulative distribution functions (CDFs) of their 30-year log returns and the (EW, ES) efficient frontiers when paired with B10 in the NN controls. As shown in Figure 5.23, the EWD (orange curve) exhibits a higher proportion of large 30-year log returns compared to the VWD (blue curve), indicating its relatively stronger return potential. Correspondingly, Figure 5.24 demonstrates that the NN control for EWD consistently outperforms VWD, yielding higher expected withdrawals across all risk levels. This superior performance aligns with EWD’s greater return potential, as reflected in its CDF, enabling more aggressive withdrawal strategies while mitigating financial risk.

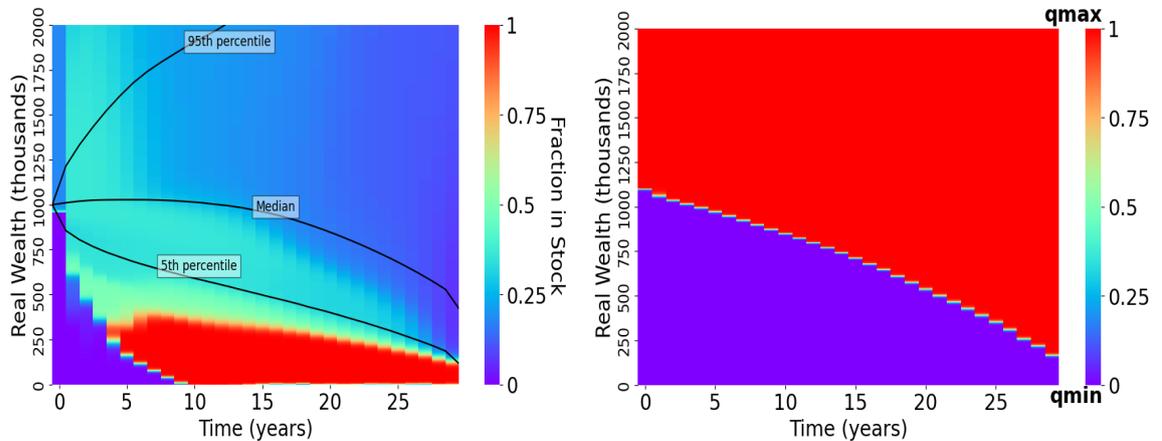
The comparison between Figures 5.25 and 5.26 reveals consistent conclusions regarding stock allocation proportions in line with prior analysis. For wealth percentiles, given that the EWD (Equal-Weighted Index) demonstrates higher returns compared to the VWD (Value-Weighted Index), the wealth curves for the EWD—across the 95th, median, and 5th percentiles—are notably higher than those of the VWD at comparable allocation levels. This disparity is particularly pronounced in the 95th percentile curve, showcasing significantly greater wealth outcomes for EWD.

In conclusion, when training the model with only two asset classes—bonds and stocks—the allocation proportion in stocks remains largely unchanged. This indicates that the allocation strategy is relatively insensitive to variations in individual asset return distributions, maintaining a consistent preference for stocks as a core portfolio component. Such stability is particularly beneficial in dynamic market environments, as it provides a reliable foundation for long-term investment performance and risk management.



(a) Fraction in stocks, NN Control, CRSP data up to 2023. (b) Normalized withdrawal, NN Control, CRSP data up to 2023.

Figure 5.25: Fraction in stocks heatmap and wealth percentiles, and normalized withdrawal, with controls computed from Problem (2.18). Note: investment setup is as in Table 5.1. Hyperparameters setup in Table 5.2. NN solution performance computed on  $2.56 \times 10^5$  observations of bootstrap resampling CPI adjusted CRSP historical data (CRSP 10-Year T-Bond Returns (B10) & CRSP Value-Weighted Index (VWD)) from 1926:1 to 2023:12. The wealth percentiles are calculated based on pre-withdrawal wealth  $W_t^-$ , while the asset allocation heatmaps are computed from post-withdrawal allocation decisions. Expected block size of bootstrap resampling is 3, see Table 3.2.  $\kappa = 1.0$ . Monetary units: USD\$ in thousands.



(a) Fraction in stocks, NN Control, CRSP data up to 2023. (b) Normalized withdrawal, NN Control, CRSP data up to 2023.

Figure 5.26: Fraction in stocks heatmap and wealth percentiles, and normalized withdrawal with controls computed from Problem (2.18). Note: investment setup is as in Table 5.1. Hyperparameters setup in Table 5.2. NN solution performance computed on  $2.56 \times 10^5$  observations of bootstrap resampling CPI adjusted CRSP historical data (CRSP 10-Year T-Bond Returns (B10) & CRSP Equal-Weighted Index (EWD)) from 1926:1 to 2023:12. The wealth percentiles are calculated based on pre-withdrawal wealth  $W_t^-$ , while the asset allocation heatmaps are computed from post-withdrawal allocation decisions. Expected block size of bootstrap resampling is 3, see Table 3.2.  $\kappa = 1.0$ . Monetary units: USD\$ in thousands.

**All assets:** Moving forward, we incorporate all CRSP assets described in Section 3.2 (CRSP

10-Year T-Bond Returns (B10) & CRSP 30-Day T-Bill Returns (T30) & CRSP 90-Day T-Bill Returns (T90) & CRSP Value-Weighted Index (VWD) & CRSP Equal-Weighted Index (EWD)) into the training of the NN framework to observe their individual characteristics and interactions within computed optimal portfolios. By analyzing the allocation proportions and return distributions for each asset, we aim to uncover how different assets contribute to the overall investment strategy. This approach enables a comprehensive evaluation of the model's ability for solving high-dimensional multi-assets environments and to balance risk and reward across a variety of asset classes.

Figure 5.27 illustrates the asset allocation and normalized withdrawal results when all assets are included in the NN training process. Notably, the allocation proportions for T30 and VWD remain nearly zero across all time stages, indicating that these assets are not prioritized in the optimal investment strategy under the given setup. The investment proportion in stocks (EWD) aligns with the previous heat maps (Figure 5.10, 5.21, 5.22, 5.25, 5.26). This reinforces the conclusion that the NN model exhibits robustness in allocating a stable proportion of the portfolio to stock asset class.

Based on the return CDF plots in Figures 5.18 and 5.23, it is evident that the returns of T30 consistently remain below those of T90, and the returns of VWD consistently remain below those of EWD. This observation aligns with the heat maps in Figure 5.27, where the allocation proportions for T30 and VWD are zero across all time horizons. The consistently lower returns make these assets less favorable for allocation under the given investment strategy.

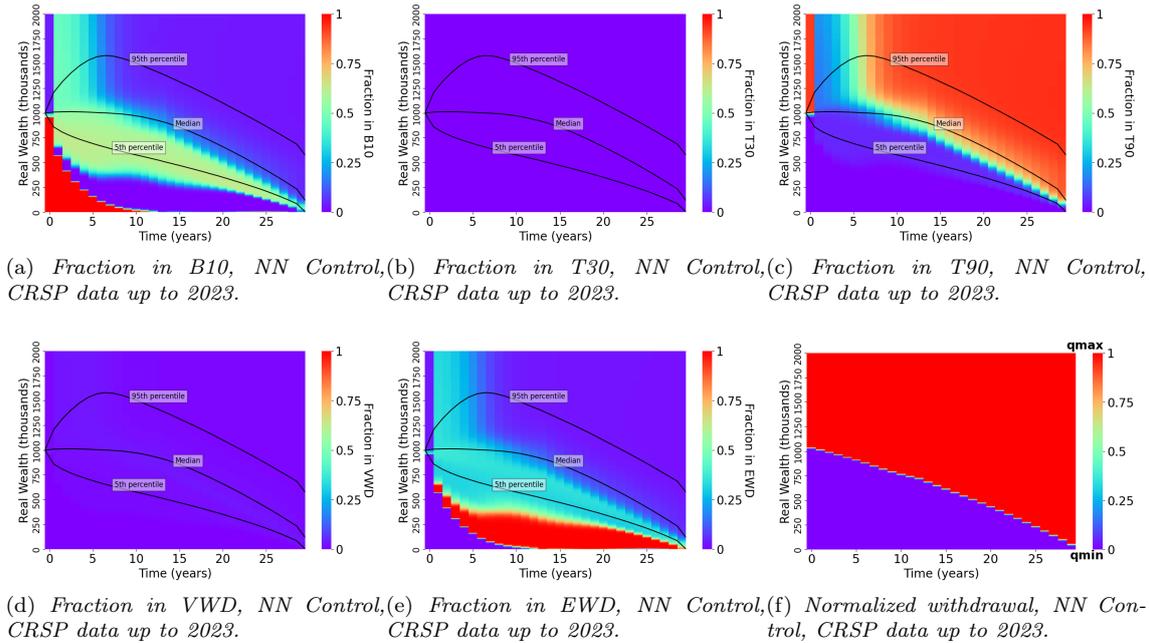


Figure 5.27: Fraction in assets heatmaps and wealth percentiles, and normalized withdrawal, with controls computed from Problem (2.18). Note: investment setup is as in Table 5.1. Hyperparameters setup in Table 5.2. NN solution performance computed on  $2.56 \times 10^5$  observations of bootstrap resampling CPI adjusted CRSP historical data (CRSP 10-Year T-Bond Returns (B10) & CRSP 30-Day T-Bill Returns (T30) & CRSP 90-Day T-Bill Returns (T90) & CRSP Value-Weighted Index (VWD) & CRSP Equal-Weighted Index (EWD)) from 1926:1 to 2023:12. The wealth percentiles are calculated based on pre-withdrawal wealth  $W_t^-$ , while the asset allocation heatmaps are computed from post-withdrawal allocation decisions. Expected block size of bootstrap resampling is 3, see Table 3.2.  $\kappa = 1.0$ . Monetary units: USD\$ in thousands.

Figure 5.28 illustrates the (EW, ES) efficient frontier obtained from training the NN framework with different asset combinations. Notably, the results from training with all CRSP assets (blue) closely resemble those from using only B10, T90, and EWD (green). This suggests that the frontier

performance is primarily driven by these three assets, consistent with the heat maps in Figure 5.27, which show near-zero allocation to T30 and VWD. However, compared to training with only two assets, such as B10 and EWD (orange), the multi-asset approach yields a more efficient result, demonstrating enhanced performance. Furthermore, the inclusion of T90 significantly improves the low-risk region, underscoring its complementary role in strengthening portfolio performance under heightened risk conditions.

In real-world scenarios, investors typically seek a balanced trade-off between rewards and risks, rather than prioritizing one at the expense of the other. This often leads to investment decisions positioned near the *knee* of the efficient frontier, where  $\kappa$  approximates 1.0 in Problem (2.18). As shown in Figure 5.28, the three performance curves around  $\kappa = 1.0$  converge closely, indicating that in this region, different asset combinations yield similar outcomes.

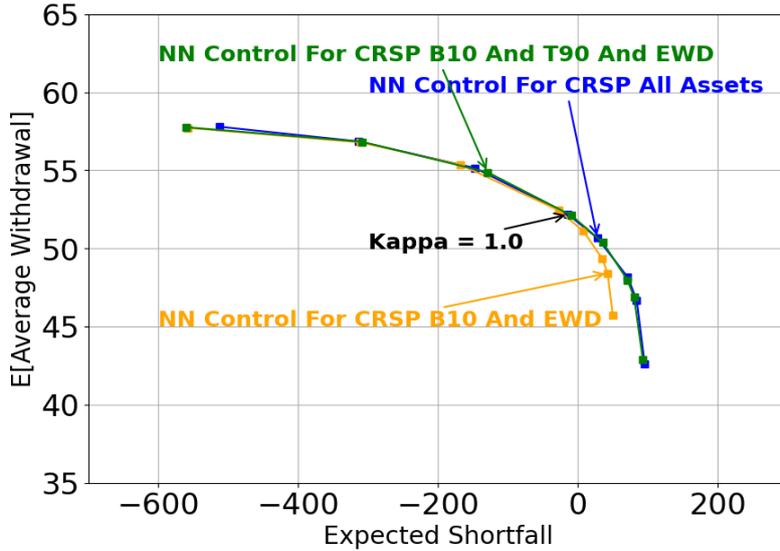


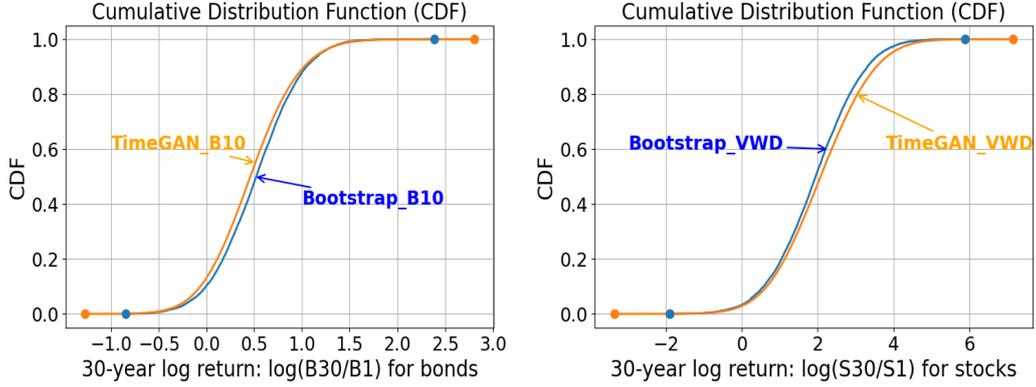
Figure 5.28: Comparison of (EW, ES) efficient frontiers for the Neural Network (NN) training for different CRSP multi-assets combinations: i) CRSP 10-Year T-Bond Returns (B10) & CRSP Equal-Weighted Index (EWD), ii) CRSP 10-Year T-Bond Returns (B10) & CRSP 90-Day T-Bill Returns (T90) & CRSP Equal-Weighted Index (EWD), iii) CRSP 10-Year T-Bond Returns (B10) & CRSP 30-Day T-Bill Returns (T30) & CRSP 90-Day T-Bill Returns (T90) & CRSP Value-Weighted Index (VWD) & CRSP Equal-Weighted Index (EWD), computed from Problem (2.18). Note: investment setup in Table 5.1. Hyper-parameters setup in Table 5.2. Control computed from the NN model, trained on  $2.56 \times 10^5$  observations of bootstrapping resampling CPI adjusted CRSP historical data from 1926:1 to 2023:12. Expected block size of bootstrap resampling is 3, see Table 3.2.

## 5.6 Computational Investigation in TimeGAN Synthetic Data Generation

In Section 3.5, we introduced the TimeGAN framework as one of our data generation techniques. In this section, we present key findings from our analysis of TimeGAN.

### 5.6.1 TimeGAN Synthetic Data Generating Performance

To evaluate the performance of TimeGAN in generating CPI adjusted CRSP historical data (CRSP 10-Year T-Bond Returns & CRSP Value-Weighted Index, from 1926:1 to 2023:12), we compare the cumulative distribution functions (CDFs) of 30-year log returns between the Bootstrapping Resampling historical data and the TimeGAN-generated data.



(a) CRSP 10-Year T-Bond Returns (B10) 30-year log-return cdf plot (b) CRSP Value-Weighted Index (VWD) 30-year log-return cdf plot

Figure 5.29: Comparison of the cumulative distribution functions (CDF) of 30-year log returns between the bootstrapping resampling historical data and the TimeGAN-generated historical data (CPI adjusted CRSP 10-Year T-Bond Returns (B10) & CRSP Value-Weighted Index (VWD), from 1926:1 to 2023:12). The data sets are generated using  $2.56 \times 10^5$  observations of data for each curve. Expected block size of bootstrap resampling is 3, see Table 3.2. TimeGAN hyperparameter settings are presented in Table 3.4, with varying  $(\lambda, \eta)$  combinations.

The 30-year log return distributions presented in Figures 5.29 (a) and 5.29 (b) indicate that the synthetic returns generated by TimeGAN for B10 and VWD closely align with those obtained through bootstrap resampling. This demonstrates the strong data generation capability of the TimeGAN framework in capturing the temporal properties of historical returns.

However, when comparing the tail distributions (i.e., the extreme values at both ends of the CDF) of the cumulative distribution functions (CDFs) for the bootstrap resampling data and the TimeGAN-generated data, we observe that the TimeGAN data exhibits heavier tails in both B10 and VWD assets. To further investigate these extreme events, we apply the threshold method proposed by Cont and Mancini (2011); Mancini (2009); Dang and Forsyth (2016) to detect jumps<sup>1</sup> in both bootstrap resampled data set and TimeGAN-generated data set.

We use the following rule to define a *jump*:

$$\left| \Delta \hat{R}_g \right| > 3\hat{\sigma} \frac{\sqrt{\Delta h}}{(\Delta h)^{0.015}}, \quad (5.8)$$

where  $\Delta h = 1$  for each month,  $\Delta \hat{R}_g$  is the de-trended monthly log return at month  $g$ , and  $\hat{\sigma}$  is the estimated standard deviation of the de-trended monthly returns over the entire dataset.

The de-trended log return  $\Delta \hat{R}_g$  is calculated as

$$\Delta \hat{R}_g = \Delta R_g - \hat{m}\Delta h, \quad (5.9)$$

where the raw log return  $\Delta R_g$  (see Section 5.4.1) is defined as

$$\Delta R_g = \log \left( \frac{P_{g+1}}{P_g} \right) \quad (5.10)$$

and the drift  $\hat{m}$  (see Section 3.3) is estimated by

$$\hat{m} = \frac{\log(P_{end}) - \log(P_{start})}{12T} \quad (5.11)$$

<sup>1</sup>In financial time series, a *jump* refers to a sudden and significant change in asset price that cannot be explained by normal market volatility. Such events are often modeled using jump diffusion processes (refer to Section 3.3), which incorporate both continuous price movements and discrete jumps, capturing extreme market behavior like crashes or surges.

where  $P_g$  denotes the asset price at month  $g$ ,  $P_{end}$  is the asset price at the end of the sample period,  $P_{start}$  is the asset price at the start of the sample period,  $T$  is the total length of the sample period (i.e., 30 years in our case), and  $\Delta h$  is the time increment (i.e., 1 month in our case). This de-trending adjustment removes the long-term trend to better isolate short-term deviations such as jumps

For both the bootstrap resampled data and the TimeGAN-generated data, we have a total of  $2.56 \times 10^5$  observations of 30-year monthly real returns for the B10 and VWD assets, based on CPI-adjusted CRSP historical data from 1926:1 to 2023:12. Using the threshold method (5.8), we count the number of jumps for each asset over a 30-year time span. The results show that, on average, there are 6 (B10) and 5 (VWD) jumps in the bootstrap resampled dataset, compared to 3 (B10) and 4 (VWD) jumps in the TimeGAN-generated dataset.

These findings suggest that while using a uniform distribution as the random input to TimeGAN (see Section 3.5) enables the model to generate jump-like behavior, it does not accurately replicate the frequency of real market jumps. The under-representation of jumps in the synthetic data indicates that a uniform latent space may fail to effectively capture rare and extreme events.

Interestingly, despite having fewer detected jumps, the 30-year log return CDFs from the TimeGAN-generated data exhibit heavier tails than those from the bootstrap resampled data (see Figure 5.29). We conjecture that, this apparent inconsistency may be explained by the fact that TimeGAN-generated jumps, although less frequent, tend to be more extreme in magnitude, thereby contributing disproportionately to the tails of the return distribution. In contrast, the bootstrap method preserves a greater number of historical jumps, but those jumps are generally smaller and less impactful in terms of tail behavior. As a result, TimeGAN underestimates jump frequency but may overestimate tail risk due to the exaggerated severity of synthetic jumps.

However, it is worth noting that for the non-jump components of the time series, the TimeGAN framework with the uniform distribution as the random input performs reasonably well. The TimeGAN-generated data closely matches the bootstrap resampled data in terms of general temporal dynamics (see Figure 5.29). This suggests that although uniform inputs may not be ideal for modeling tail events, they are still capable of preserving the overall structure and time-dependent properties of typical market behavior.

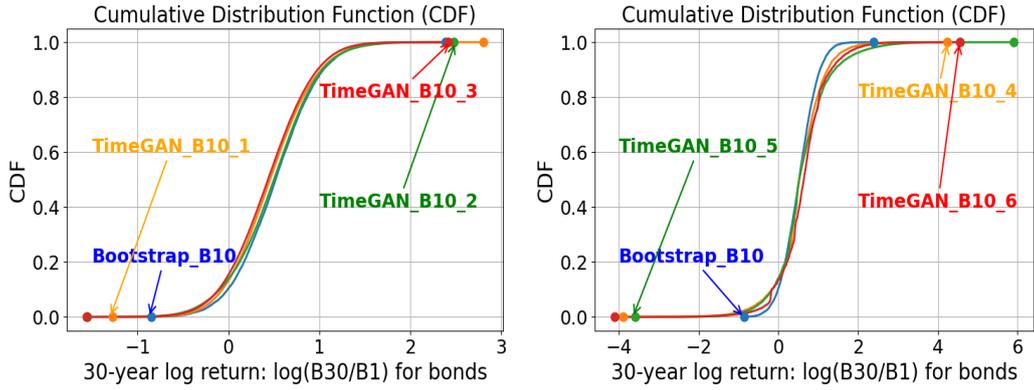
## 5.6.2 Investigation for TimeGAN Hyper-parameter Settings

In this section, we explore the impact of adjusting various hyper-parameter settings on the performance of TimeGAN-generated data to assess whether these modifications influence its data generation quality.

### Loss Balancing Hyper-parameter

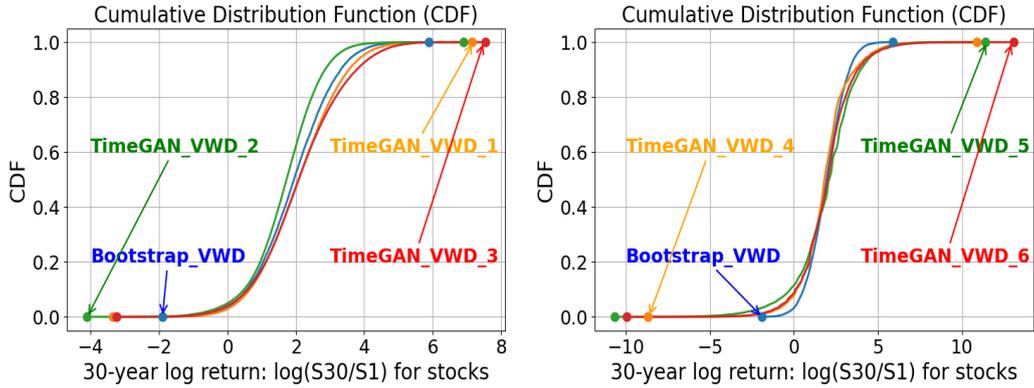
In Section 3.5.1, two loss balancing hyper-parameters are introduced:  $\lambda$  in equation (3.12) and  $\eta$  in (3.13). While Yoon et al. (2019) suggests that TimeGAN is not particularly sensitive to these hyper-parameters, we conduct experiments to assess their impact on TimeGAN’s training performance for our CRSP historical dataset.

We evaluate six different combinations of the loss balancing hyperparameters  $(\lambda, \eta)$ : (0.1, 100), (0.5, 50), (1, 10), (10, 1), (50, 0.5), and (100, 0.1). To analyze their impact, we compare the cumulative distribution functions (CDFs) of 30-year log returns generated using TimeGAN, based on CPI-adjusted CRSP 10-Year T-Bond Returns and CRSP Value-Weighted Index from 1926:1 to 2023:12. The TimeGAN training follows the hyperparameter settings specified in Table 3.4.



(a) CRSP 10-Year T-Bond Returns (B10) 30-year log-return cdf plot (b) CRSP 10-Year T-Bond Returns (B10) 30-year log-return cdf plot

Figure 5.30: Comparison of the cumulative distribution functions (CDF) of 30-year log returns between the bootstrapping resampling historical data and the TimeGAN-generated historical data (CPI adjusted CRSP 10-Year T-Bond Returns (B10), from 1926:1 to 2023:12). The data sets are generated using  $2.56 \times 10^5$  observations of data for each curve. Expected block size of bootstrap resampling is 3, see Table 3.2. TimeGAN hyper-parameter settings are presented in Table 3.4, with varying  $(\lambda, \eta)$  combinations. 1:(0.1,100), 2:(0.5,50), 3:(1,10), 4:(10,1), 5:(50,0.5), 6:(100,0.1)



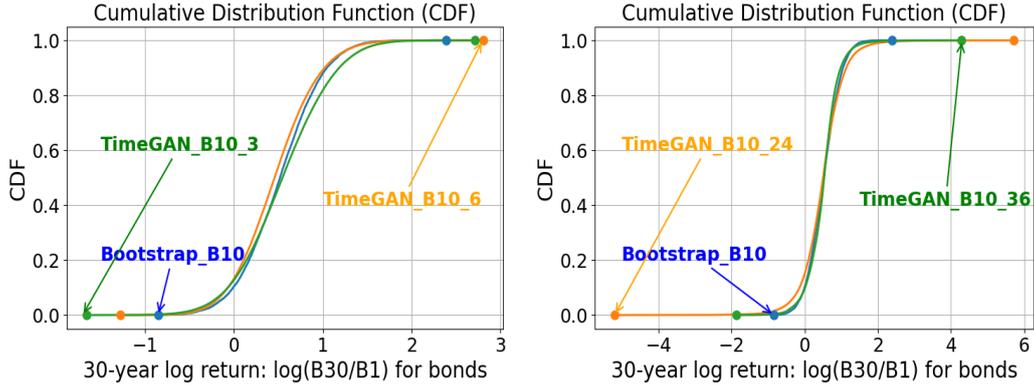
(a) CRSP Value-Weighted Index (VWD) 30-year log-return cdf plot (b) CRSP Value-Weighted Index (VWD) 30-year log-return cdf plot

Figure 5.31: Comparison of the cumulative distribution functions (CDF) of 30-year log returns between the bootstrapping resampling historical data and the TimeGAN-generated historical data (CPI adjusted CRSP Value-Weighted Index (VWD), from 1926:1 to 2023:12). The data sets are generated using  $2.56 \times 10^5$  observations of data for each curve. Expected block size of bootstrap resampling is 3, see Table 3.2. TimeGAN hyper-parameter settings are presented in Table 3.4, with varying  $(\lambda, \eta)$  combinations. 1:(0.1,100), 2:(0.5,50), 3:(1,10), 4:(10,1), 5:(50,0.5), 6:(100,0.1)

Figures 5.30 and 5.31 illustrate that, for our CRSP historical data, TimeGAN generates broadly similar results across different hyperparameter settings, though some differences remain. Using the Bootstrap Resampling results as a benchmark, we observe that when  $\lambda > \eta$  (Figure 5.30 (b) and Figure 5.31 (b)), the performance of TimeGAN deteriorates compared to cases where  $\lambda < \eta$  (Figure 5.30 (a) and Figure 5.31 (a)). Additionally, the distributions of TimeGAN-generated datasets with  $\lambda > \eta$  exhibit greater dispersion than the corresponding bootstrap resampling distributions for both bond and equity returns. Among the six tested  $(\lambda, \eta)$  combinations, the setting (0.1, 100) yields the best performance, as its generated distributions most closely align with the Bootstrap Resampling benchmark.

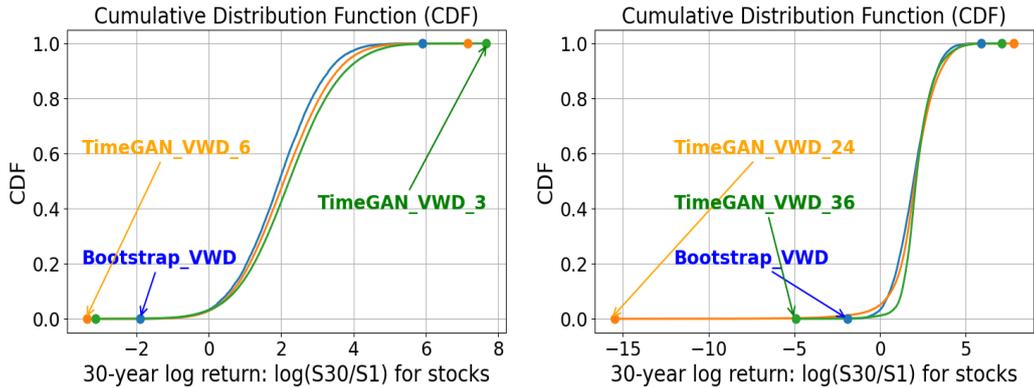
## Sequence Length

Previously, we set the training sequence length to 6, referred to [Staden et al. \(2023\)](#) (see Table 3.4). In this section, we examine the impact of sequence length on time series training using TimeGAN. Specifically, we evaluate the model’s performance across four different sequence lengths: 3, 6, 24, and 36 months, corresponding to the block size in bootstrap resampling method, see Section 3.4. We still use the same CPI adjusted CRSP data (CRSP 10-Year T-Bond Returns and CRSP Value-Weighted Index from 1926:1 to 2023:12) as the TimeGAN training source data and conduct the cumulative distribution functions (CDFs) of 30-year log returns.



(a) CRSP 10-Year T-Bond Returns (B10) 30-year log-return cdf plot (b) CRSP 10-Year T-Bond Returns (B10) 30-year log-return cdf plot

Figure 5.32: Comparison of the cumulative distribution functions (CDF) of 30-year log returns between the bootstrapping resampling historical data and the TimeGAN-generated historical data (CPI adjusted CRSP 10-Year T-Bond Returns (B10), from 1926:1 to 2023:12). The data sets are generated using  $2.56 \times 10^5$  observations of data for each curve. Expected block size of bootstrap resampling is 3, see Table 3.2. TimeGAN hyper-parameter settings are presented in Table 3.4, with varying sequence lengths for 3 months, and 6 months.



(a) CRSP Value-Weighted Index (VWD) 30-year log-return cdf plot (b) CRSP Value-Weighted Index (VWD) 30-year log-return cdf plot

Figure 5.33: Comparison of the cumulative distribution functions (CDF) of 30-year log returns between the bootstrapping resampling historical data and the TimeGAN-generated historical data (CPI adjusted CRSP Value-Weighted Index (VWD), from 1926:1 to 2023:12). The data sets are generated using  $2.56 \times 10^5$  observations of data for each curve. Expected block size of bootstrap resampling is 3, see Table 3.2. TimeGAN hyper-parameter settings are presented in Table 3.4, with varying sequence lengths for 24 months, and 36 months.

Figures 5.32 and 5.33 show that, when using bootstrap resampling results as the benchmark, the tail distributions for sequence lengths of 3 and 6 months align more closely with the benchmark compared to those for sequence lengths of 24 and 36 months. This suggests that when longer sequence lengths (24 or 36 months) are used as training data in the TimeGAN model, the model tends to generate more exaggerated extreme data than shorter sequence lengths (3 or 6 months). It also indicates that the sequence length—corresponding to the block size in bootstrap resampling—plays a crucial role: the closer the sequence length is to the expected block size of bootstrap resampling (which is optimal to be 3 in our CRSP historical data case, see Table 3.2), the more accurately the TimeGAN model can replicate results consistent with bootstrap resampling.

# Chapter 6

## Model Robustness

A common potential pitfall of neural networks is over-fitting to the training data. Neural networks that are over-fitted do not have the ability to generalize to unseen data. Since future asset return paths cannot be predicted, it is important to ascertain that the computed strategy is not overfitted to the training data and can perform well on unseen return paths. In this chapter, we demonstrate the robustness of the NN model's generated controls.

### 6.1 CRSP Bootstrap Resampling Historical Data Testing

To ensure the robustness of the model, we conduct tests using Bootstrapping Resampling CPI adjusted CRSP historical data (CRSP 10-Year T-Bond Returns (B10) & CRSP Value-Weighted Index (VWD)) from different time periods (1926:1 - 2019:12 & 1926:1 - 2023:12). Any historical dataset named *new historical data* means Bootstrapping Resampling CRSP historical data spanning from 1926:1 to 2023:12; any historical dataset named *old historical data* means Bootstrapping Resampling CRSP historical data spanning from 1926:1 to 2019:12. By testing the model on varying datasets, we aim to evaluate the robustness of the model's performance across diverse market conditions.

We conduct three types of robustness tests: (i) out-of-sample testing, (ii) out-of-distribution testing, and (iii) control sensitivity to training distribution.

#### 6.1.1 Out-of-sample testing

Out-of-sample tests in Figure 6.1 involve testing model performance on an unseen data set sampled from the same distribution. In our case, this means training the NN on one set of return paths sampled from the 1926:1 - 2023:12 historical data, and testing on another set of paths generated using a different random seed.

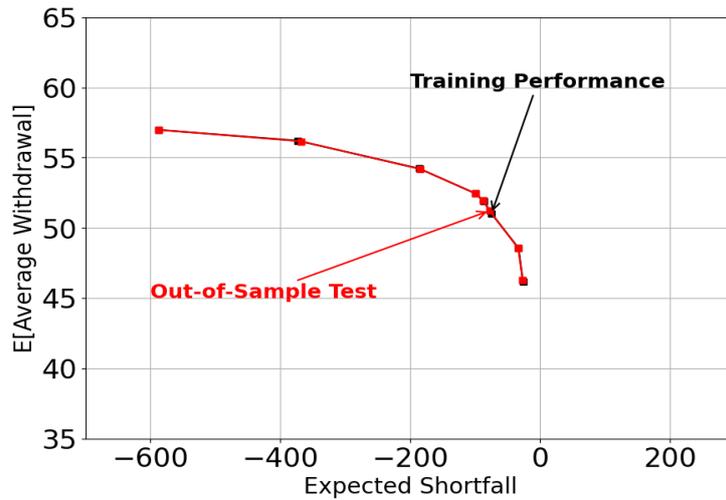


Figure 6.1: *Out-of-sample test. Comparison of NN training performance results vs. out-of-sample test using (EW, ES) efficient frontiers. Both training and testing datasets are  $2.56 \times 10^5$  observations of new historical data (1926:1 - 2023:12), generated with different random seeds. Note: investment setup in Table 5.1. Hyper-parameters setup in Table 5.2.*

### 6.1.2 Out-of-distribution testing

Out-of-distribution testing involves evaluating the performance of the computed control on a data set sampled from a different distribution. Specifically, (i) The training (EW, ES) efficient frontier would be on the new historical dataset, with testing (EW, ES) efficient frontier on model trained on old historical data on new data (Figure 6.2). (ii) The training (EW, ES) efficient frontier would be on the old historical dataset, with testing (EW, ES) efficient frontier on model trained on new historical data on old data (Figure 6.3).

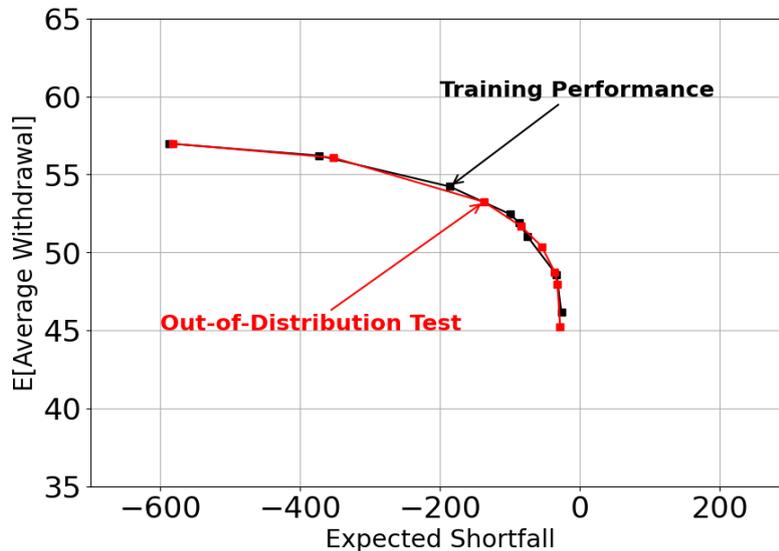


Figure 6.2: *Out-of-distribution test. Comparison of NN training performance results vs. out-of-distribution test using (EW, ES) efficient frontier. The training (EW, ES) efficient frontier is on the new historical dataset (1926:1 - 2023:12), with testing (EW, ES) efficient frontier on model trained on old historical data (1926:1 - 2019:12) on new data. Both training and testing data sets contain  $2.56 \times 10^5$  observations. Note: investment setup in Table 5.1. Hyper-parameters setup in Table 5.2.*

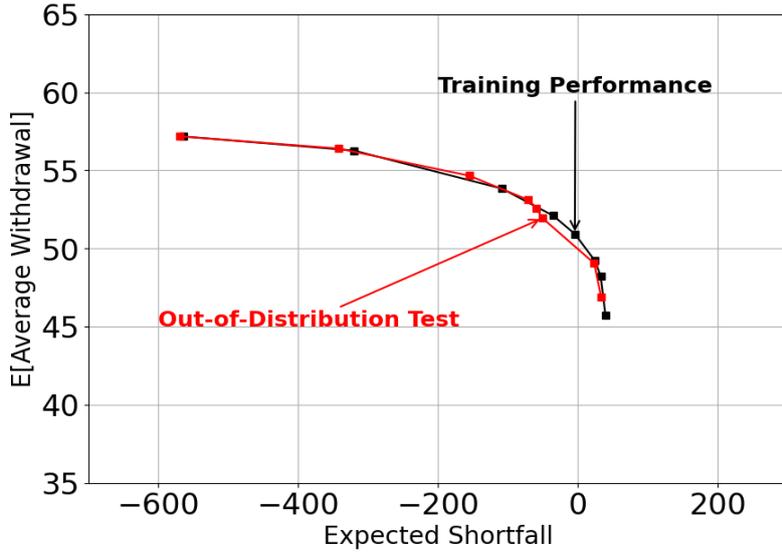
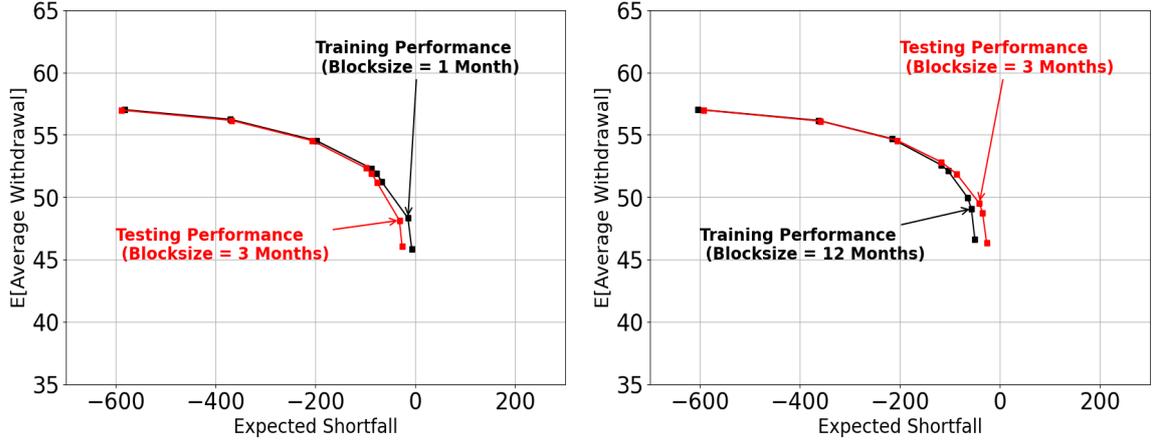


Figure 6.3: *Out-of-distribution test. Comparison of NN training performance results vs. out-of-distribution test using (EW, ES) efficient frontier. The training (EW, ES) efficient frontier is on the old historical dataset (1926:1 - 2019:12), with testing (EW, ES) efficient frontier on model trained on new historical data (1926:1 - 2023:12) on old data. Both training and testing data sets contain  $2.56 \times 10^5$  observations. Note: investment setup in Table 5.1. Hyper-parameters setup in Table 5.2.*

From Figure 6.2, it can be observed that if we apply the NN model trained on the old historical data to the new historical data (red), the resulting test outcomes align closely with the results obtained by directly training on the new historical data (black). From Figure 6.3, it can be observed that if we apply the model trained on the new historical data to the old historical data (red), the resulting test outcomes align closely with the results obtained by directly training on the old historical data (black). These two figures demonstrate that even when training with different data distributions, the resulting model remains applicable to other data distributions. Specifically, we can use the trained NN model parameters generated from the existing data to provide recommendations for future investment allocations based on the nice performance in Figure 6.2.

### 6.1.3 Sensitivity to training distribution

To test the NN framework’s sensitivity to training data set, we train the NN framework on new historical data with expected block sizes (see description in section 3.4) of both 1 month and 12 months and then test the resulting control on new historical data with expected block size of 3. See testing result in Figure 6.4.



(a) Historical training data, block size = 1 month, NN Control (b) Historical training data, block size = 12 months, NN Control

Figure 6.4: Training on new historical data. (EW, ES) efficient frontiers of controls generated by NN model trained on  $2.56 \times 10^5$  observations of new historical data (1926:1 - 2023:12) with expected block sizes of a) 1 month and b) 12 months, each tested on  $2.56 \times 10^5$  observations of new historical data with expected block sizes of 3 months. Note: investment setup in Table 5.1. Hyper-parameters setup in Table 5.2.

## 6.2 TimeGAN Data Testing

To further test the robustness of the NN control trained on CRSP historical data, we introduce TimeGAN-generated data for additional testing. The TimeGAN synthetic data is generated from applying TimeGAN technique to CPI adjusted CRSP historical data (CRSP 10-Year T-Bond Returns & CRSP Value-Weighted Index), from 1926:1 to 2023:12, using hyper-parameter settings in Table 3.4. And the resampled data comes from bootstrapping resampling CPI adjusted CRSP historical data (CRSP 10-Year T-Bond Returns & CRSP Value-Weighted Index), from 1926:1 to 2023:12, using expected block size of 3. This allows evaluation of the model's consistency and adaptability under both real and synthetic data constructions. Specifically, (i) The training (EW, ES) efficient frontier would be on the TimeGAN dataset, with testing (EW, ES) efficient frontier on Bootstrapping Resampling historical dataset ((EW, ES) efficient frontier in Figure 6.5, heat maps in Figure 6.6). (ii) The training (EW, ES) efficient frontier would be on the Bootstrapping Resampling historical dataset, with testing (EW, ES) efficient frontier on the TimeGAN dataset ((EW, ES) efficient frontier in Figure 6.7, heat maps in Figure 6.8).

The results from Figures 6.5 and 6.7 demonstrate that the NN control exhibits strong robustness across different datasets. The (EW, ES) efficient frontiers obtained from training on real historical data and TimeGAN-generated data are highly similar, indicating that the learned strategy generalizes well across both real and synthetic environments. This suggests that the NN control is not overly sensitive to specific characteristics of the training data and can maintain consistent performance under varying data distributions.

The heat maps presented in Figure 6.6 and Figure 6.8 exhibit consistent patterns with the findings from the analysis in Section 5.3, demonstrating that our neural network (NN) framework possesses strong robustness in dynamic investment strategy.

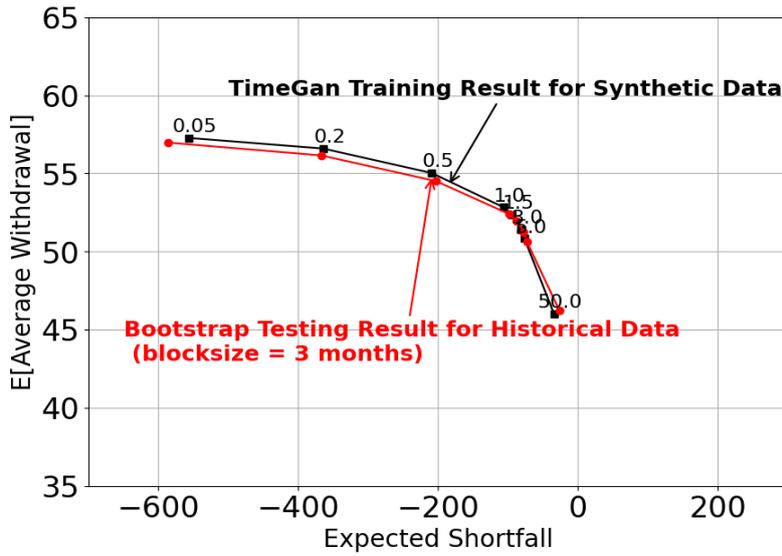
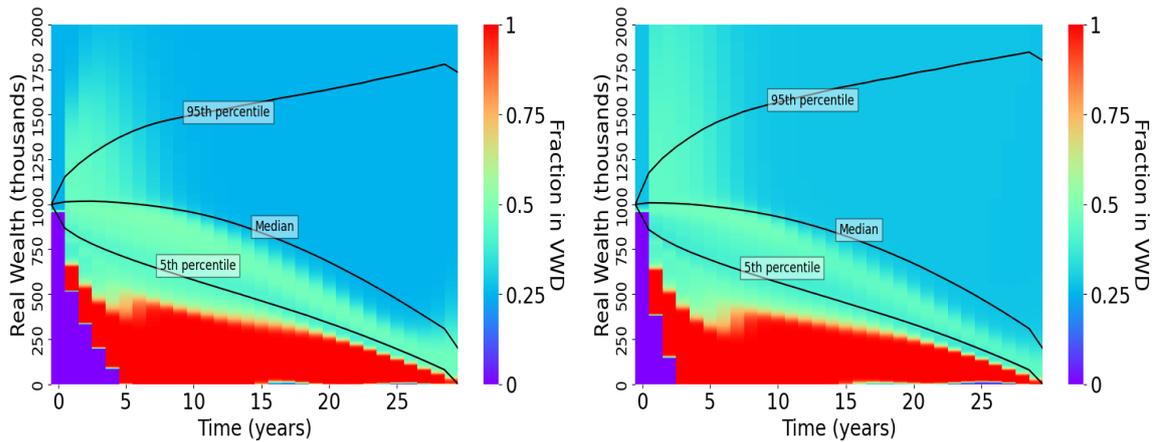


Figure 6.5:  $(EW, ES)$  efficient frontiers of controls generated by NN model trained on  $2.56 \times 10^5$  observations of TimeGAN data, tested on  $2.56 \times 10^5$  observations of bootstrapping resampling historical data from 1926:1 to 2023:12. Note: investment setup in Table 5.1. Hyper-parameters setup in Table 5.2.



(a) Fraction in stocks, NN Control, NN training performance. (b) Fraction in stocks, NN Control, NN testing performance.

Figure 6.6: Fraction in stocks heatmap and wealth percentiles generated by NN model trained on  $2.56 \times 10^5$  observations of TimeGAN data, tested on  $2.56 \times 10^5$  observations of bootstrapping resampling historical data from 1926:1 to 2023:12. Note: investment setup in Table 5.1. Hyper-parameters setup in Table 5.2. The wealth percentiles are calculated based on pre-withdrawal wealth  $W_t^-$ , while the asset allocation heatmaps are computed from post-withdrawal allocation decisions.  $\kappa = 1.0$ . Monetary units: USD\$ in thousands.

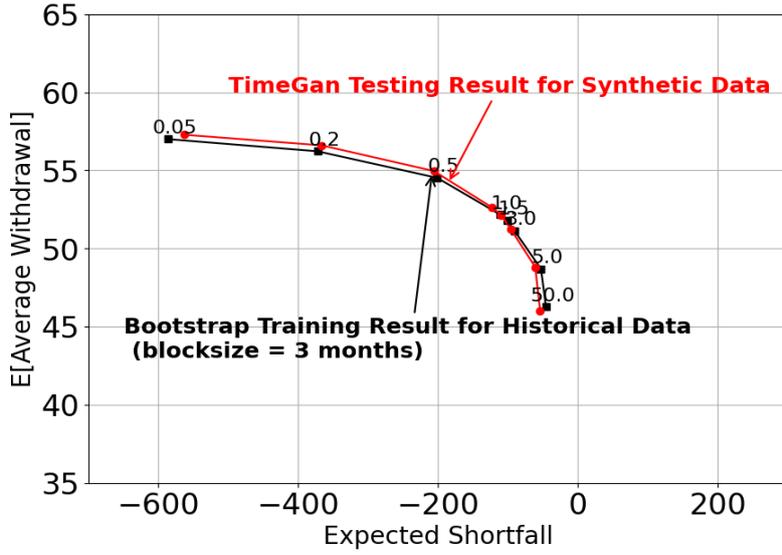
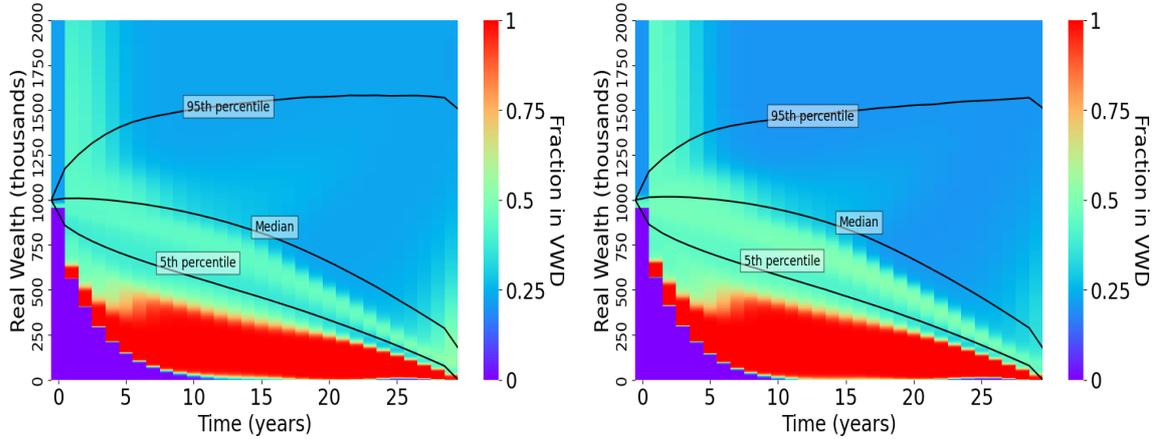


Figure 6.7:  $(EW, ES)$  efficient frontiers of controls generated by NN model trained on  $2.56 \times 10^5$  observations of bootstrapping resampling historical data from 1926:1 to 2023:12, tested on  $2.56 \times 10^5$  observations of TimeGAN data. Note: investment setup in Table 5.1. Hyper-parameters setup in Table 5.2.



(a) Fraction in stocks, NN Control, NN training performance. (b) Fraction in stocks, NN Control, NN testing performance.

Figure 6.8: Fraction in stocks heatmap and wealth percentiles generated by NN model trained on  $2.56 \times 10^5$  observations of bootstrapping resampling historical data from 1926:1 to 2023:12, tested on  $2.56 \times 10^5$  observations of TimeGAN data. Note: investment setup in Table 5.1. Hyper-parameters setup in Table 5.2. The wealth percentiles are calculated based on pre-withdrawal wealth  $W_t^-$ , while the asset allocation heatmaps are computed from post-withdrawal allocation decisions.  $\kappa = 1.0$ . Monetary units: USD\$ in thousands.

# Chapter 7

## Conclusion

In this thesis, we provide a summary of traditional approaches to the Optimal Dynamic Allocation Problem, conduct a comprehensive review of the NN Control method proposed in [Chen et al. \(2023\)](#), and carry out further investigation into this framework.

The main contributions of this study are fourfold: (1) developing and evaluating training strategies, particularly transfer learning, to address data scarcity issues in low-risk regions caused by the inherent challenges of risk minimization; (2) assessing the neural network framework’s scalability in high-dimensional spaces through comprehensive analysis of multi-asset training outcomes; (3) implementing and investigating the TimeGAN data generation technique, including an in-depth exploration of its training performance; and (4) conducting rigorous robustness testing of the neural network framework across diverse asset classes.

These contributions are realized through the following key innovations:

- **Non-Parametric, Data-Driven Optimization** Unlike the HJB equation approach, which relies on parametric stochastic models (e.g., jump-diffusion processes requiring calibrated parameters in [Table 3.1](#)), the NN framework operates without explicit assumptions about asset return dynamics. By directly learning control strategies from sample return trajectories, the NN approach eliminates model misspecification risks and bypasses the need for complex SDE calibrations.
- **Computational Accuracy** The NN framework achieves numerical precision comparable to the HJB benchmark. By leveraging transfer learning, we resolve training challenges in risk-sensitive regions, enabling accurate approximation of constrained stochastic controls.
- **Computational Robustness** The NN framework demonstrates consistent performance across diverse market conditions, maintaining strategic stability through rigorous out-of-sample and out-of-distribution evaluations. Heatmap analyses confirm negligible variation in allocation patterns, with robust adaptability to both historical and synthetic financial environments, effectively mitigating overfitting risks in non-stationary markets.
- **High-Dimensional Scalability** The NN approach successfully addresses the dimensionality challenges inherent in multi-asset portfolio optimization. By directly learning control strategies from data, the NN framework overcomes the curse of dimensionality, without relying on computationally intensive conditional expectation calculations. Comparative analysis of cumulative distribution functions (CDFs) further validates its capacity to handle complex asset interactions while maintaining Pareto-optimal outcomes.
- **TimeGAN-Enhanced Synthetic Data Generation** We employ TimeGAN to synthesize realistic financial time-series data that preserves the temporal dynamics and cross-asset correlations inherent in CRSP historical datasets. By learning the latent structures of market returns, the framework generates synthetic trajectories that replicate key temporal properties. A systematic investigation of TimeGAN hyperparameters—such as sequence length and loss function configurations—reveals critical insights into TimeGAN training performance. The synthesized data further validates the neural network model’s robustness.

This thesis demonstrates how neural networks can transform the way we approach complex financial challenges in retirement planning. By moving beyond traditional models that rely on rigid assumptions, the framework developed here adapts directly to real-world data, offering solutions that are both precise and practical.

The success of the NN framework lies in its ability to learn from market behavior without being tied to predefined equations or idealized scenarios. Whether handling multiple asset classes or navigating rare market crises, the model maintains stability and coherence—proving that data-driven strategies can outperform conventional methods constrained by theoretical limitations.

Ultimately, this research shifts the focus from mathematical abstraction to real-world adaptability, providing a foundation for retirement strategies that evolve with markets and empower individuals to face financial uncertainty with confidence.

# References

- Bender, J., Briand, R., Melas, D., and Subramanian, R. A. (2013). Foundations of factor investing. Available at SSRN 2543990.
- Bengen, W. (1994). Determining withdrawal rates using historical data. *Journal of Financial Planning*, 7:171–180.
- Bernhardt, T. and Donnelly, C. (2018). Pension decumulation strategies: A state of the art report. Technical report, Risk Insight Lab, Heriot Watt University.
- Buehler, H., Gonon, L., Teichmann, J., and Wood, B. (2019). Deep hedging. *Quantitative Finance*, 19(8):1271–1291.
- Center for Research in Security Prices (CRSP) (Accessed 2024). CRSP US Stock and Bond Databases. University of Chicago Booth School of Business. Data accessed via Wharton Research Data Services (WRDS), Wharton School, University of Pennsylvania. Available at: <https://wrds-www.wharton.upenn.edu> and <https://www.crsp.org>.
- Chen, M., Shirazi, M., Forsyth, P. A., and Li, Y. (2023). Machine learning and hamilton-jacobi-bellman equation for optimal decumulation: a comparison study. *arXiv*, 2306.10582.
- Cont, R. and Mancini, C. (2011). Nonparametric tests for pathwise properties of semimartingales. *Bernoulli*, 17:781–813.
- Dang, D.-M. and Forsyth, P. A. (2016). Better than pre-commitment mean-variance portfolio allocation strategies: a semi-self-financing hamilton-jacobi-bellman equation approach. *European Journal of Operational Research*, 250:827–841.
- Dichtl, H., Drobetz, W., and Wambach, M. (2016). Testing rebalancing strategies for stock-bond portfolios across different asset allocations. *Applied Economics*, 48(9):772–788.
- Forsyth, P. A. (2020). Multi-period mean cvar asset allocation: Is it advantageous to be time consistent? *SIAM Journal on Financial Mathematics*, 11(2):358–384.
- Forsyth, P. A. (2022). A stochastic control approach to defined contribution plan decumulation: "the nastiest, hardest problem in finance". *North American Actuarial Journal*, 26(2):227–251.
- Forsyth, P. A. and Labahn, G. (2019).  $\epsilon$ -monotone fourier methods for optimal stochastic control in finance. *Journal of Computational Finance*, 22(4):25–71.
- Forsyth, P. A. and Vetzal, K. R. (2019). Optimal asset allocation for retirement savings: deterministic vs. time consistent adaptive strategies. *Applied Mathematical Finance*, 26(1):1–37.
- Forsyth, P. A., Vetzal, K. R., and Westmacott, G. (2022). Optimal performance of a tontine overlay subject to withdrawal constraints. *arXiv*, 2211.10509.
- Forsyth, P. A., Vetzal, K. R., and Westmacott, G. (2024). Optimal performance of a tontine overlay subject to withdrawal constraints. *ASTIN Bulletin: The Journal of the IAA*, 54:94–128.
- Han, J. and E., W. (2016). Deep learning approximation for stochastic control problems. *CoRR*, abs/1611.07422.

- Huré, C., Pham, H., Bachouch, A., and Langrené, N. (2021). Deep neural networks algorithms for stochastic control problems on finite horizon: Convergence analysis. *SIAM Journal on Numerical Analysis*, 59(1):525–557.
- Johnson, O. V., Xinying, C., Khaw, K. W., and Lee, M. H. (2023). ps-calr: Periodic-shift cosine annealing learning rate for deep neural networks. *IEEE Access*, 11.
- Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*.
- Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Kou, S. G. (2002). A jump-diffusion model for option pricing. *Management Science*, 48:1086–1101.
- Kou, S. G. and Wang, H. (2004). Option pricing under a double exponential jump diffusion model. *Management Science*, 50:1178–1192.
- Li, Y. and Forsyth, P. A. (2019). A data-driven neural network approach to optimal asset allocation for target based defined contribution pension plans. *Insurance: Mathematics and Economics*, 86:189–204.
- Lin, Y., MacMinn, R., and Tian, R. (2015). De-risking defined benefit plans. *Insurance: Mathematics and Economics*, 63:52–65.
- Loshchilov, I. and Hutter, F. (2016). Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint, arXiv:1608.03983*.
- MacMinn, R., Brockett, P., Wang, J., Lin, Y., and Tian, R. (2014). The securitization of longevity risk and its implications for retirement security. In Mitchell, O. S., Maurer, R., and Hammond, P. B., editors, *Recreating Sustainable Retirement*, pages 134–160. Oxford University Press, Oxford.
- Mancini, C. (2009). Non-parametric threshold estimation models with stochastic diffusion coefficient and jumps. *Scandinavian Journal of Statistics*, 36:270–296.
- Marler, R. T. and Arora, J. S. (2004). Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395.
- Ni, C., Li, Y., Forsyth, P., and Carroll, R. (2022). Optimal asset allocation for outperforming a stochastic benchmark target. *Quantitative Finance*, 22(8):1421–1440.
- Patton, A., Politis, D., and White, H. (2009). Correction to: automatic block-length selection for the dependent bootstrap. *Econometric Reviews*, 28:372–375.
- Pfau, W. D. (2018). An overview of retirement income planning. *Journal of Financial Counseling and Planning*, 29(1):114–120.
- Pfeiffer, S., Salter, J. R., and Evensky, H. E. (2013). Increasing the sustainable withdrawal rate using the standby reverse mortgage. *Journal of Financial Planning*, 26(12):55–62.
- Politis, D. and Romano, J. (1994). The stationary bootstrap. *Journal of the American Statistical Association*, 89:1303–1313.
- Politis, D. and White, H. (2004). Automatic block-length selection for the dependent bootstrap. *Econometric Reviews*, 23:53–70.
- Ritholz, B. (2017). Tackling the ‘nastiest, hardest problem in finance’. <https://www.bloomberg.com/view/articles/2017-06-05/tackling-the-nastiest-hardest-problem-in-finance>.
- Rockafellar, R. T. and Uryasev, S. (2000). Optimization of conditional value-at-risk. *Journal of Risk*, 2:21–42.

- Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.
- Shefrin, H. M. and Thaler, R. H. (1988). The behavioral life-cycle hypothesis. *Economic Inquiry*, 26(4):609–643.
- Smith, L. N. (2017). Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472. IEEE.
- Smith, S., Elsen, E., and De, S. (2020). On the generalization benefit of noise in stochastic gradient descent. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 9058–9067. PMLR.
- Staden, P. V., Forsyth, P. A., and Li, Y. (2023). Beating a benchmark: dynamic programming may not be the right numerical approach. *SIAM Journal on Financial Mathematics*, 14(2):407–451.
- Strub, M., Li, D., and Cui, X. (2019). An enhanced mean-variance framework for robo-advising applications. Technical Report 3302111, SSRN. Available at SSRN: [https://papers.ssrn.com/sol3/papers.cfm?abstract\\_id=3302111](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3302111).
- Tankov, P. and Cont, R. (2009). *Financial Modelling with Jump Processes*. Chapman and Hall/CRC, New York.
- Tsang, K. H. and Wong, H. Y. (2020). Deep-learning solution to portfolio selection with serially-dependent returns. SSRN, 10.2139.
- van Staden, P. M., Forsyth, P. A., and Li, Y. (2024). A global-in-time neural network approach to dynamic portfolio optimization. Working paper, University of Waterloo.
- Williams, R. and Kawashima, C. (2023). Beyond the 4% rule: How much can you spend in retirement?
- Yoon, J., Jarrett, D., and van der Schaar, M. (2019). Time-series generative adversarial networks. In *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, pages 5509–5519.
- Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., and He, Q. (2021). A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76.