

Title	Generative Reward Machines for reinforcement learning in problems with poor observability
Authors	Rezaei, Saeid
Publication date	2026-06-01
Original Citation	Rezaei, S. 2026. Generative Reward Machines for reinforcement learning in problems with poor observability. PhD Thesis, University College Cork.
Type of publication	Doctoral thesis
Rights	© 2026, Saeid Rezaei - https://creativecommons.org/licenses/by-nc/4.0/
Download date	2026-10-25 04:31:10
Item downloaded from	https://hdl.handle.net/10468/19289

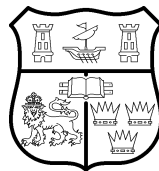
Generative Reward Machines for Reinforcement Learning in Problems with Poor Observability

Saeid Rezaei

MSc.

119221883

**Thesis submitted for the degree of
Doctor of Philosophy**



NATIONAL UNIVERSITY OF IRELAND, CORK

FACULTY OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

June 2026

Head of Department: Prof Dirk Pesch

Supervisors: Professor Kenneth N. Brown
Professor Barry O'Sullivan

Contents

List of Figures	v
List of Tables	viii
Acknowledgements	xviii
Abstract	xxi
1 Introduction	1
1.1 Motivation	1
1.2 Thesis Statement	2
1.3 What was achieved?	2
1.4 The Implementation Examples	3
1.5 How is the claim validated?	4
1.6 Who is going to use this?	4
1.7 Thesis contributions	5
1.8 Publications	6
1.9 Thesis layout	7
2 Literature review and background	8
2.1 Multi-Criteria Decision-Making in Resource Allocation	8
2.2 Reinforcement Learning	9
2.2.1 Fundamentals of Reinforcement Learning	9
2.2.2 Fundamental Concepts and Notations	10
2.2.3 Reinforcement Learning Pipeline	12
2.2.4 Q-function	14
2.2.5 Partially Observable Environments	15
2.2.6 The Belief-State	16
2.2.7 DDPG	16
2.2.8 Automated Reinforcement Learning	17
2.3 Reward Machines	19
2.3.1 Integration with MDPs	20
2.3.2 Foundational Reward Machine Algorithms	21
2.4 Real-World Challenges in Reinforcement Learning	22
2.4.1 Data-Related Challenges	23
2.4.2 Environment-Related Challenges	24
2.4.3 Operational Challenges	26
2.4.4 Real-World Reinforcement Learning benchmark	26
2.5 Extending Real-World Reinforcement Learning Benchmarks	28
2.5.1 A Quantitative Framework for Evaluating Deep Learning Training Pipelines	28
2.6 A Benchmark for Footstep Biometrics	30
2.7 The Omnigenomic Benchmark (OGB)	32
2.7.1 OGB Task Descriptions	32
2.7.2 Evaluation and Scoring Framework	33
2.8 Reconnaissance Blind Chess	34
2.9 The Physical Internet	40
2.9.1 Conceptual Foundations and Architectural Framework	40

2.9.2	Indicators of Systemic Unsustainability	41
2.9.3	Core Components and Definitions	42
2.9.4	Analogy with the Digital Internet	42
2.10	Related Work in Physical Internet Research	44
2.10.1	Architectural and Operational Frameworks	44
2.10.2	Modeling and Analysis Methodologies	45
2.10.3	Governance, Sustainability, and Implementation	45
2.11	Capacity Management in the Physical Internet	46
2.11.1	Foundational Frameworks for Capacity Management	46
2.11.2	Optimization Methods for Prescriptive Analytics	47
2.12	Conclusion	48
3	Uncertainty quantification for RL algorithms in POMDP environments	49
3.1	Introduction	49
3.2	Existence of Optimal policy	51
3.2.1	Convergence of the value function	52
3.2.2	Convergence of the Q function	52
3.2.3	Convergence of SQ Learning	53
3.2.4	Convergence of Q-value Decision Processes	53
3.2.5	Random reward functions	55
3.2.6	Random reward functions and ϵ optimal policies	57
3.3	Guide for Problem Solvability	59
3.4	Implications for Algorithm Selection and GRM	60
3.5	Conclusion	61
4	Generative reward machine for RL problems	62
4.1	Introduction	62
4.2	Generative Reward Machines	62
4.2.1	Comparison: Reward Machine vs. Generative Reward Machine	64
4.3	The GRM Framework for Reconnaissance Blind Chess	66
4.3.1	RL for RBC	67
4.3.2	The GRM Framework for RBC	68
4.3.3	Analysis of Algorithm Performance in the 2022 NeurIPS Tournament	69
4.4	The GRM Framework for Biometric Footstep Recognition	72
4.4.1	Problem modeling	72
4.4.2	RL Framework	73
4.4.3	The GRM Framework	74
4.4.4	Results	76
4.5	The GRM Framework for OGB benchmark	77
4.6	GRM Framework	78
4.6.1	Results	82
4.7	Conclusion	82
5	GymDC environment	84

5.1	Introduction	84
5.2	Distribution Center Simulator	85
5.2.1	Distribution Center System Design Overview	85
5.2.2	Distribution center modeling	86
5.2.3	Understanding Incoming Packet Parameters in Simulator Settings	89
5.2.4	Matching Algorithm	92
5.2.5	Performance Evaluation Under Uncertainty Scenarios	93
5.3	Reinforcement Learning Environment	94
5.3.1	State Space	94
5.3.2	Reward Function	95
5.3.3	Action Space	96
5.4	Performance Comparison of Baselines Across Settings	96
5.5	Simulation settings	97
5.6	Performance Analysis for Simulation Settings in different scenarios	99
5.7	Discussion	116
5.8	Deep Reinforcement Learning on Gym-DC	118
5.8.1	Scaling the Challenge: Agent Performance from easy to hard	118
5.8.2	Reinforcement Learning Agent Configuration	118
5.8.3	Results	118
5.9	Performance Analysis for Simulation Settings in different scenarios	119
5.10	Discussion	138
5.11	Conclusion	140
6	Generative reward machine for GymDC	142
6.1	Introduction	142
6.2	GRM Framework for the Gym-DC Environment	143
6.2.1	Estimator for Packet Arrival Patterns	143
6.2.2	GRM State and Reward Formulation	144
6.3	Performance Analysis for Simulation Settings in different scenarios	144
6.4	Discussion	152
6.5	Conclusion	154
7	Concluding remarks	155
7.1	Future work	156
7.1.1	Future Work in GymDC	156
7.1.2	Future Work in PI Simulation	157
7.1.3	Future Work in Enhancing the GRM	157
8	Appendices	172
8.1	Comparative Performance of Baseline Strategies	173
8.2	Comparative Analysis of the RL Agent and Baseline Strategies	177
8.3	Full Strategy Comparison for Settings 4 and 5	181

8.4	Glossary of Hyperparameters	183
8.5	Details of the Cart-Pole Swing-Up Environment	186
8.6	Strategies Employed by Tournament Bots	188
8.7	Comparative Methods	191

List of Figures

2.1	The agent-environment interaction in a Markov decision process.	10
2.2	Feature Engineering stage in the Online Pipeline.	18
2.3	Feature Engineering stage in the Offline Pipeline.	19
2.4	A visual representation of partial observability in RBC. The White player senses a 3x3 portion of the board (red rectangle) and must move without knowledge of events outside this view, such as a recent capture.	36
3.1	QDP in the perspective of other decision problem classes[MH18].	54
4.1	Bayes Elo ratings and 95% confidence intervals as the tournament progressed, excluding forfeits.[GPS ⁺ 22]	71
5.1	Physical Internet Graph	85
5.2	Conceptual diagram for DC	86
5.3	Five different distributions over 10 days, with a total of 500 packets.	90
5.4	Flow diagram from the point of view of the packet	91
5.5	Plotting performance of different baselines for Setting 1 in On-time delivery window size = 20	101
5.6	Plotting performance of different baselines for Setting 2 in On-time delivery window size = 20	102
5.7	Plotting performance of different baselines for Setting 3 in On-time delivery window size = 20	103
5.8	Plotting performance of different baselines for Setting 4 in On-time delivery window size = 20	104
5.9	Plotting performance of different baselines for Setting 5 in On-time delivery window size = 20	105
5.10	Plotting performance of different baselines for Setting 1 in Random delay window size = 20	106
5.11	Plotting performance of different baselines for Setting 2 in Random delay window size = 20	107
5.12	Plotting performance of different baselines for Setting 3 in Random delay window size = 20	108
5.13	Plotting performance of different baselines for Setting 4 in Random delay window size = 20	109
5.14	Plotting performance of different baselines for Setting 5 in Random delay window size = 20	110
5.15	Plotting performance of different baselines for Setting 1 in Unreliable service provider window size = 20	111
5.16	Plotting performance of different baselines for Setting 2 in Unreliable service provider window size = 20	112
5.17	Plotting performance of different baselines for Setting 3 in Unreliable service provider window size = 20	113

5.18	Plotting performance of different baselines for Setting 4 in Unreliable service provider window size = 20	114
5.19	Plotting performance of different baselines for Setting 5 in Unreliable service provider window size = 20	115
5.20	Plotting performance of different baselines for Setting1 in On-time delivery window size = 20	123
5.21	Plotting performance of different baselines for Setting1 in Random delay window size = 20	124
5.22	Plotting performance of different baselines for Setting1 in Unreliable service provider window size = 20	125
5.23	Plotting performance of different baselines for Setting2 in On-time delivery window size = 20	126
5.24	Plotting performance of different baselines for Setting2 in Random delay window size = 20	127
5.25	Plotting performance of different baselines for Setting2 in Unreliable service provider window size = 20	128
5.26	Plotting performance of different baselines for Setting3 in On-time delivery window size = 20	129
5.27	Plotting performance of different baselines for Setting3 in Random delay window size = 20	130
5.28	Plotting performance of different baselines for Setting3 in Unreliable service provider window size = 20	131
5.29	Plotting performance of different baselines for Setting4 in On-time delivery window size = 20	132
5.30	Plotting performance of different baselines for Setting4 in Random delay window size = 20	133
5.31	Plotting performance of different baselines for Setting4 in Unreliable service provider window size = 20	134
5.32	Plotting performance of different baselines for Setting5 in On-time delivery window size = 20	135
5.33	Plotting performance of different baselines for Setting5 in Random delay window size = 20	136
5.34	Plotting performance of different baselines for Setting5 in Unreliable service provider window size = 20	137
6.1	Plotting performance of different baselines for Setting 4 in On-time delivery window size = 20	146
6.2	Plotting performance of different baselines for Setting 4 in Random delay window size = 20	147
6.3	Plotting performance of different baselines for Setting 4 in Unreliable service provider window size = 20	148
6.4	Plotting performance of different baselines for Setting 5 in On-time delivery window size = 20	149
6.5	Plotting performance of different baselines for Setting 5 in Random delay window size = 20	150

6.6	Plotting performance of different baselines for Setting 5 in Unreliable service provider window size = 20	151
-----	---	-----

List of Tables

2.1	Notations	11
2.2	Fundamental definitions in Reinforcement Learning	13
2.3	The RL environment settings for each challenge[DALM ⁺ 21] . .	27
2.4	A Comparative Analysis of Game Mechanics: Classical Chess vs. Reconnaissance Blind Chess.	38
2.5	OLI model	43
3.1	Structured Approach to Developing Reinforcement Learning Applications	50
4.1	Comparison of RM and GRM Components	65
4.2	Comparison of Average Elo scores between the RL algorithm and GRM framework in the tournament, excluding forfeits.[GPS ⁺ 22]	71
4.3	Comparison of Weighted Total scores between the RL and GRM on the Biometric Footstep Recognition Benchmark	77
4.4	Comparison of performance metrics for different teams in International StepUP Competition for Biometric Footstep Recognition.	77
4.5	Comparison of Weighted Total scores between the RL algorithm and GRM framework on the OGB Benchmark	82
4.6	Comparison of Weighted Total for different frameworks on the OGB Benchmark.	82
5.1	Important simulator parameters	88
5.2	Comparison of Parameters across Different settings	98
5.3	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 1, On-time delivery.	101
5.4	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 1 in On-time delivery.	101
5.5	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 2, On-time delivery.	102
5.6	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 2 in On-time delivery.	102
5.7	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 3, On-time delivery.	103
5.8	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 3 in On-time delivery.	103
5.9	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 4, On-time delivery.	104
5.10	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 4 in On-time delivery.	104
5.11	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 5, On-time delivery.	105
5.12	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 5 in On-time delivery.	105

5.13	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 1, Random delay.	106
5.14	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 1 in Random delay.	106
5.15	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 2, Random delay.	107
5.16	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 2 in Random delay.	107
5.17	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 3, Random delay.	108
5.18	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 3 in Random delay.	108
5.19	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 4, Random delay.	109
5.20	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 4 in Random delay.	109
5.21	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 5, Random delay.	110
5.22	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 5 in Random delay.	110
5.23	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 1, Unreliable service provider.	111
5.24	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 1 in Unreliable service provider.	111
5.25	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 2, Unreliable service provider.	112
5.26	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 2 in Unreliable service provider.	112
5.27	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 3, Unreliable service provider.	113
5.28	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 3 in Unreliable service provider.	113
5.29	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 4, Unreliable service provider.	114
5.30	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 4 in Unreliable service provider.	114
5.31	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 5, Unreliable service provider.	115
5.32	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 5 in Unreliable service provider.	115
5.33	Frequency of Baselines Achieving Maximum Performance. The table displays the number of times each baseline achieved the highest performance value out of 5000 runs for each scenario and setting. Note that multiple baselines can simultaneously achieve the maximum performance in a single run.	117
5.34	RL Agent Specifications	119

5.35	Average 1000-Episode Reward Across Different Complexity Levels.	119
5.36	RL Agent and Environment Specifications	120
5.37	Server Hardware Specifications	121
5.38	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting1, On-time delivery. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j. .	123
5.39	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting1 in On-time delivery. Cells represent the number of seeds with higher values for the row baseline.	123
5.40	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting1, Random delay. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j. .	124
5.41	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting1 in Random delay. Cells represent the number of seeds with higher values for the row baseline.	124
5.42	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting1, Unreliable service provider. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.	125
5.43	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting1 in Unreliable service provider. Cells represent the number of seeds with higher values for the row baseline.	125
5.44	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting2, On-time delivery. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j. .	126
5.45	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting2 in On-time delivery. Cells represent the number of seeds with higher values for the row baseline.	126
5.46	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting2, Random delay. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j. .	127
5.47	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting2 in Random delay. Cells represent the number of seeds with higher values for the row baseline.	127
5.48	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting2, Unreliable service provider. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.	128

5.49	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting2 in Unreliable service provider. Cells represent the number of seeds with higher values for the row baseline.	128
5.50	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting3, On-time delivery. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j. .	129
5.51	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting3 in On-time delivery. Cells represent the number of seeds with higher values for the row baseline.	129
5.52	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting3, Random delay. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j. .	130
5.53	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting3 in Random delay. Cells represent the number of seeds with higher values for the row baseline.	130
5.54	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting3, Unreliable service provider. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.	131
5.55	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting3 in Unreliable service provider. Cells represent the number of seeds with higher values for the row baseline.	131
5.56	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting4, On-time delivery. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j. .	132
5.57	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting4 in On-time delivery. Cells represent the number of seeds with higher values for the row baseline.	132
5.58	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting4, Random delay. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j. .	133
5.59	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting4 in Random delay. Cells represent the number of seeds with higher values for the row baseline.	133
5.60	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting4, Unreliable service provider. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.	134

5.61	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting4 in Unreliable service provider. Cells represent the number of seeds with higher values for the row baseline.	134
5.62	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting5, On-time delivery. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j. .	135
5.63	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting5 in On-time delivery. Cells represent the number of seeds with higher values for the row baseline.	135
5.64	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting5, Random delay. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j. .	136
5.65	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting5 in Random delay. Cells represent the number of seeds with higher values for the row baseline.	136
5.66	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting5, Unreliable service provider. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.	137
5.67	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting5 in Unreliable service provider. Cells represent the number of seeds with higher values for the row baseline.	137
5.68	Classification of Environment Settings by Complexity Level. . .	138
5.69	Performance Comparison of Baselines and RL Agent. The table shows the frequency, out of 5000 runs, that each strategy attained the maximum performance score for a given experimental setting.	139
6.1	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 4, On-time delivery.	146
6.2	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 4 in On-time delivery.	146
6.3	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 4, Random delay.	147
6.4	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 4 in Random delay.	147
6.5	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 4, Unreliable service provider.	148
6.6	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 4 in Unreliable service provider.	148

6.7	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 5, On-time delivery.	149
6.8	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 5 in On-time delivery.	149
6.9	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 5, Random delay.	150
6.10	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 5 in Random delay.	150
6.11	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 5, Unreliable service provider.	151
6.12	Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 5 in Unreliable service provider.	151
6.13	Comparative Performance of RL and Generative Agents Against Baselines. The table shows the frequency (out of 5000 runs) that each strategy achieved the maximum performance score.	153
8.1	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 1, On-time delivery.	173
8.2	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 2, On-time delivery.	173
8.3	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 3, On-time delivery.	174
8.4	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 4, On-time delivery.	174
8.5	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 5, On-time delivery.	174
8.6	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 1, Random delay.	174
8.7	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 2, Random delay.	174
8.8	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 3, Random delay.	175
8.9	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 4, Random delay.	175
8.10	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 5, Random delay.	175
8.11	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 1, Unreliable service provider.	175
8.12	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 2, Unreliable service provider.	175
8.13	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 3, Unreliable service provider.	176
8.14	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 4, Unreliable service provider.	176
8.15	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 5, Unreliable service provider.	176

8.16	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 1, On-time delivery. Each cell $[i][j]$ represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j . .	177
8.17	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 1, Random delay. Each cell $[i][j]$ represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j . .	177
8.18	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 1, Unreliable service provider. Each cell $[i][j]$ represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j	177
8.19	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 2, On-time delivery. Each cell $[i][j]$ represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j . .	178
8.20	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 2, Random delay. Each cell $[i][j]$ represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j . .	178
8.21	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 2, Unreliable service provider. Each cell $[i][j]$ represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j	178
8.22	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 3, On-time delivery. Each cell $[i][j]$ represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j . .	178
8.23	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 3, Random delay. Each cell $[i][j]$ represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j . .	179
8.24	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 3, Unreliable service provider. Each cell $[i][j]$ represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j	179
8.25	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting4, On-time delivery. Each cell $[i][j]$ represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j . .	179

8.26	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting4, Random delay. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j. .	179
8.27	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting4, Unreliable service provider. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.	180
8.28	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting5, On-time delivery. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j. .	180
8.29	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting5, Random delay. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j. .	180
8.30	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting5, Unreliable service provider. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.	180
8.31	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 4, On-time delivery.	181
8.32	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 4, Random delay.	181
8.33	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 4, Unreliable service provider.	181
8.34	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 5, On-time delivery.	181
8.35	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 5, Random delay.	182
8.36	Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 5, Unreliable service provider.	182
8.37	Agent Behavioral Hyperparameters	183

I, Saeid Rezaei, certify that this thesis is my own work and has not been submitted for another degree at University College Cork or elsewhere.

Saeid Rezaei

Dedicated to my beloved parents who have sacrificed so much for my
education.

Acknowledgements

I wish to express my sincere gratitude to my primary supervisor, Professor Kenneth Brown, for his invaluable support, guidance, and intellectual mentorship throughout my research career. His expertise proved instrumental not only in the successful preparation and refinement of this thesis but also in significantly contributing to my professional development.

I also extend my thanks to my secondary supervisor, Professor Barry O’Sullivan.

I would like to thank Linda O’Sullivan and Caitriona Walsh for their support in the computer science lab. Finally, a special thank you to Peter J. MacHale for his prompt technical help with the servers and computing infrastructure that made this project possible.

Funding Acknowledgement

This research was supported by Science Foundation Ireland under Grant numbers 16/RC/3918 and 12/RC/2289-P2 which are co-funded under the European Regional Development Fund.

*All models are approximations.
Essentially, all models are
wrong, but some are useful.
However, the approximate
nature of the model must always
be borne in mind.*

George Box
(1919–2013)

Abstract

Reinforcement learning provides methodologies for sequential decision-making in autonomous systems. While reinforcement learning has achieved success in fully observable environments, real-world applications frequently suffer from partial observability, system delays, and non-stationary dynamics. For example, autonomous robots performing specific tasks and systems managing complex logistics networks often face these constraints. The dynamic and noisy nature of these practical applications causes standard reinforcement learning algorithms to fail.

The first question addressed in this research explores how specific environment conditions can render reinforcement learning problems fundamentally not solvable. While other research classifies problem complexity into easy, medium, and hard categories based merely on adjusting environment settings, this work takes a theoretical approach. It reviews specific conditions, demonstrating through mathematical proofs that optimal and epsilon-optimal policies cannot exist when environments feature changing reward functions and aleatoric uncertainty. By identifying the underlying issues that make a problem not solvable, this research explains why certain environments become truly hard for standard algorithms. This theoretical perspective acts as a guide to determine whether a decision-making problem can be solved or whether it requires specialized approaches.

To address these unsolvable problems and environments with obstructed state visibility, this research proposes the Generative Reward Machine framework. Unlike Reward Machines that rely on fully observable environments, the Generative Reward Machine utilizes an estimator function to transform noisy, partial observations into a stable state representation. This generative mechanism allows the system to establish stationary reward signals despite environmental uncertainty. By creating this intermediate state space, the system can develop policies even when the true state of the environment remains hidden.

The efficacy of the Generative Reward Machine framework is empirically validated across diverse domains. The first evaluations involve Reconnaissance Blind Chess, a game characterized by severe information constraints, where the framework adapted strategic hyperparameters to classify opponent skill levels. Further validations applied the framework to hyperparameter optimization for biometric footprint recognition and the Omnigenomic Benchmark. In these ap-

plications, the Generative Reward Machine provided a stable learning signal, enabling the framework to achieve top-ranking performance.

The final question addressed in this research evaluates whether the Generative Reward Machine can improve capacity management within Physical Internet distribution centers. To test this, the research introduces Gym-DC, a simulation environment that models physical packet routing under varying levels of stochasticity, including random delays and unreliable service providers. While standard deep reinforcement learning agents experience severe performance degradation in the hardest Gym-DC settings, the proposed Generative Reward Machine-driven agent maintains robustness. The framework outperforms standard reinforcement learning approaches and heuristic baselines. Ultimately, this research demonstrates that integrating generative state abstractions can improve performance in stochastic routing problems in Physical Internet.

Chapter 1

Introduction

1.1 Motivation

Reinforcement learning (RL) has proven to be highly successful across a wide range of sequential decision-making problems. Recent breakthroughs have demonstrated its efficacy in complex domains, ranging from mastering competitive board games to enabling sophisticated robotic manipulation and autonomous navigation tasks. However, the foundational theory of reinforcement learning relies on assumptions regarding the problem structure, specifically state observability and a stationary reward function. In many real-world applications, these conditions simply do not hold. [DALM⁺21] formulated a series of practical challenges—such as partial observability, non-stationarity, and system delays—that must be addressed to extend the range and viability of RL techniques. In this thesis, we focus on problems characterized by poor observability and unstable reward functions.

To ground this research, we utilize the Physical Internet (PI) as our primary application domain. The PI is a paradigm-shifting proposal for an open, global logistics system driven by physical, digital, and operational interconnectivity. Within this framework, autonomous agents are required to make localized, real-time decisions regarding packet routing and transport selection. This creates an extraordinarily large and complex operational environment fraught with highly volatile and uncertain conditions. One important sub-problem within this ecosystem is the management of PI distribution centers, or hubs. In these centers, an autonomous agent must dynamically decide whether to accept incoming packets and subsequently determine exactly when, where, and how to

forward them through the network to optimize efficiency. Throughout this thesis, we employ this Distribution Center decision problem as our example.

To assist RL agents in navigating complex domains, [IKVM22] proposed Reward Machines (RMs). RMs provide a structured methodology in which the reward function is encoded as a finite state automaton operating on an extended state space. By exposing the underlying logic of the reward to the agent, this method allows a problem to be reformulated into one that maintains the critical Markovian property, improving sample efficiency. However, the standard RM relies on the assumption that the agent operates within a stable, deterministic, and fully observable environment. In complex, noisy environments like the PI distribution center, the true state is often hidden by poor observations and the underlying Markov property is lost, rendering standard reward machines inadequate.

To overcome these limitations and effectively bridge the gap between theoretical RL frameworks and real-world implementation, this research centers on the development of the Generative Reward Machine (GRM). The main research question we focus on in this thesis is:

How can Reinforcement Learning be adapted to handle environments with poor observability and unstable reward functions?

1.2 Thesis Statement

A Generative Reward Machine improves Reinforcement Learning performance on complex problems with poor observability and dynamic reward functions. Specifically:

- The reward structure for RL agents can be designed effectively based on the classification of POMDP to improve performance in some complex problems with different levels of observability.
- RL augmented with a generative reward function can improve reinforcement learning performance in high-complexity routing problems within Physical Internet distribution centers.

1.3 What was achieved?

The achievements of this research are:

- Proofs establishing conditions where reinforcement learning fails to converge to optimal or ϵ -optimal policies. The analysis examines environments with non-stationary reward functions and aleatoric uncertainty, providing justification for the Generative Reward Machine.
- Design of the Generative Reward Machine (GRM). The framework extends Reward Machines for environments with noise and partial observability. It generates state representations for policy development when observation of states is obstructed.
- Demonstration and evaluation of the GRM approach applied to four different application domains. The methodology includes implementation of a new simulation benchmark Gym-DC, which models routing and decision making in Physical Internet hubs.

1.4 The Implementation Examples

The methods in this thesis were applied to four different domains:

- **Physical Internet Hub environment (Gym-DC):** Gym-DC was developed for autonomous decision-making in a simulated Physical Internet logistics hub. This environment includes settings with high levels of change and randomness, which are known to be difficult for standard RL agents.
- **Reconnaissance Blind Chess (RBC):** The GRM framework was used to create an agent for Reconnaissance Blind Chess (RBC). The agent adapts its strategy during the game by classifying its opponent's skill level and choosing the suitable hyperparameters. Its performance was evaluated in open competition at Machine Reconnaissance Blind Chess Tournament of NeurIPS 2022.
- **Biometric Footstep Recognition:** The framework was used for Hyperparameters Optimization, which is treated as an RL problem. Its performance was evaluated in "International StepUP Competition for Biometric Footstep Recognition" in the International Joint Conference on Biometrics 2025.
- **Foundation Models for Genomic Modeling and Understanding:** The framework was used for Hyperparameters Optimization, which is treated as an RL problem. Its performance was evaluated in "Foundation Models for Genomic Modeling and Understanding" in ECML 2025.

1.5 How is the claim validated?

The research claim is validated using several methods:

- **Formal Proofs:** Mathematical proofs are used to validate the theoretical claims about the non-existence of optimal and ϵ -optimal policies under specific reward conditions.
- **Empirical Benchmarking:** The performance of the developed RL and GRM agents is compared against multiple heuristic-based policies. This comparison uses 5000 different simulation runs for each configuration to ensure the results are statistically robust.
- **Comparative Analysis:** A direct performance comparison is made between the GRM agent and a standard deep RL agent, especially in the more difficult ("hard") simulation settings, to demonstrate the GRM's performance.
- **Performance in External Competition:** The RBC agent's performance was validated through its participation in the NeurIPS tournament, which provided a competitive benchmark against other algorithms. Furthermore, the GRM-based agent achieved first place in two recent competitions: the "International StepUP Competition for Biometric Footstep Recognition" at the 2025 International Joint Conference on Biometrics (IJCB) and the "Foundation Models for Genomic Modeling" challenge at the 2025 European Conference on Machine Learning (ECML).

1.6 Who is going to use this?

This research can be used by the following groups:

- **Reinforcement Learning Researchers:** RL researchers can use the theoretical findings, the new complexity classification, and the GRM framework for work on partially observable and non-stationary problems.
- **Logistics and Supply Chain Innovators:** Professionals in the logistics industry can use the findings as a guide for designing autonomous management systems for Physical Internet hubs and other complex logistical networks.

1.7 Thesis contributions

To address these questions, this thesis makes several contributions to reinforcement learning and its application within the logistics domain:

- **The Generative Reward Machine (GRM) Framework:** We introduce the Generative Reward Machine (GRM). The GRM functions by generating abstract state representations. The mechanism enables agents to develop policies when observation of environmental states is obstructed.
- **Theoretical Foundations:** The thesis provides theorems demonstrating the conditions under which optimal and ϵ -optimal policies cannot exist. The proofs focus on scenarios characterized by non-stationary reward functions and aleatoric uncertainty. The theorems serve as a guide for when the GRM approach is required.
- **Empirical Validation:** The efficacy of the GRM framework is validated through empirical assessments across multiple domains:
 1. **Physical Internet Logistics (Gym-DC):** We have developed Gym-DC, a specialized simulation environment that accurately models Physical Internet (PI) hub operations. Experimental results demonstrate that the GRM-driven agent outperforms traditional RL methods, especially in noisy logistical environments.
 2. **Reconnaissance-Blind Chess (Imperfect Information Games):** The GRM framework is employed to dynamically refine agent strategies in the domain of Reconnaissance-Blind Chess. It accomplishes this by effectively classifying opponent skill levels and adaptively selecting optimal behavioral hyperparameters, thereby proving its effectiveness within competitive, partially observable contexts.
 3. **Biometric Footstep Recognition and Foundation Models for Genomic Modeling and Understanding (Adaptive Hyperparameter Optimization):** GRM is leveraged to enhance the efficiency of meta-learning, specifically for hyperparameter optimization in deep learning. Empirical results demonstrate improved performance on two benchmarks: the Omnigenomic Benchmark (OGB) and footstep recognition benchmark.

1.8 Publications

Most of the work presented in this thesis has been published or has received awards in competitions. These are detailed below.

1. **“Gym-DC: A Distribution Centre Reinforcement Learning Environment”** Saeid Rezaei, Federico Toffano, and Kenneth N. Brown. . In: Pattern Recognition, Computer Vision, and Image Processing. ICPR 2022 International Workshops and Challenges. ICPR 2022. Lecture Notes in Computer Science, vol 13644. Springer, Cham. Springer, 2023, pp. 687–699. DOI: 10.1007/978-3-031-37742-6_53.
https://doi.org/10.1007/978-3-031-37742-6_53
2. **“Generative Reward Machine for Reinforcement Learning for Physical Internet Distribution Centre”** Saeid Rezaei and Kenneth N. Brown. . In: Machine Learning, Optimization, and Data Science: 10th International Conference, LOD 2024, Castiglione della Pescaia, Italy, September 22–25, 2024, Revised Selected Papers, Part I, pp. 317–332. DOI: 10.1007/978-3-031-82481-4_22.
https://dl.acm.org/doi/10.1007/978-3-031-82481-4_22
3. **“The Machine Reconnaissance Blind Chess Tournament of NeurIPS 2022”** Ryan W. Gardner, Gino Perrotta, Anvay Shah, Shivaram Kalyanakrishnan, Kevin A. Wang, Gregory Clark, Timo Bertram, Johannes Fürnkranz, Martin Müller, Brady P. Garrison, Prithviraj Dasgupta, and Saeid Rezaei. . In: Proceedings of the NeurIPS 2022 Competitions Track , Proceedings of Machine Learning Research , 2022 ,PMLR , 119–132.
<https://proceedings.mlr.press/v220/gardner23a.html>
4. **“First International StepUP Competition for Biometric Footstep Recognition: Methods, Results and Remaining Challenges”** Robyn Larracy, Eve MacDonald, Angkoon Phinyomark, Saeid Rezaei, Mahdi Laghaei, Ali Hajighasem, Aaron Tabor, and Erik Scheme. In: 2025 IEEE International Joint Conference on Biometrics ,.
<https://ijcb2025.ieee-biometrics.org/competitions/>
5. **“ECML Discovery Challenge on Foundation Models for Genomic Modeling and Understanding”** In: 2025 European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases - ECML PKDD , September 2025 ,Porto, Portugal.

<https://www.codabench.org/competitions/6930/>

1.9 Thesis layout

The remainder of this thesis is structured as follows:

Chapter 2 presents foundational concepts in the Physical Internet, reinforcement learning, and real-world challenges encountered in reinforcement learning, followed by illustrative examples such as reconnaissance-blind chess and hyperparameter optimization in deep learning models.

Chapter 3 presents the theoretical contributions of the thesis. It details the conditions for the convergence of RL algorithms, introduces the new complexity classification framework, and provides formal proofs for the non-existence of optimal policies in certain environments.

Chapter 4 introduces the Generative Reward Machine (GRM) framework. It details its implementation and evaluates its performance on the complex problem domains of Reconnaissance Blind Chess, Omnigenomic Benchmark (OGB) and a footstep recognition benchmark.

Chapter 5 introduces the Gym-DC simulation environment, detailing the five distinct operational settings and three scenarios designed to test RL agents under varying levels of complexity and stochasticity. Furthermore, the chapter evaluates the performance of a standard deep RL agent against several heuristic policies across different settings of the Gym-DC environment.

Chapter 6 describes GRM for the Gym-DC Environment. This chapter applies the GRM framework to the challenging settings of the Gym-DC environment, providing a comparative analysis against the standard RL agent to demonstrate the GRM's effectiveness in highly stochastic conditions.

Finally, potential future directions for this work are described in Chapter 7.

Chapter 2

Literature review and background

2.1 Multi-Criteria Decision-Making in Resource Allocation

In numerous operational and strategic scenarios, a fundamental challenge lies in the optimal allocation or matching of entities from two distinct groups. This requirement has propelled the development and application of sophisticated matching algorithms. For instance, in the domain of human resource management, one might seek to assign employees to projects based on their skills and project requirements. Similarly, in transportation logistics, carriers are matched with shipments considering factors such as cost, capacity, and delivery timelines. A particularly salient application is found within capacity management problems. In this context, one set of entities can be conceptualized as items requiring storage or processing, such as packages in a warehouse, while the other set comprises service providers or storage locations, each with varying capabilities and costs. The objective is to establish a matching that optimizes one or more performance metrics, a task for which multi-criteria decision-making (MCDM) algorithms are exceptionally well-suited.

MCDM encompasses a suite of analytical methods designed to evaluate and select among a set of alternatives based on multiple, often conflicting, criteria. The core principle of many MCDM techniques is to quantify the performance of each alternative with respect to each criterion and then to aggregate these performances into a single comprehensive score or ranking. An illustrative and widely adopted MCDM method is the Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS). The fundamental concept of TOPSIS is

to identify an ideal solution, which represents the best possible performance across all criteria, and a negative-ideal solution, which embodies the worst performance. As detailed in a recent comprehensive review, the algorithm then determines the preference order of alternatives by simultaneously measuring the Euclidean distance of each alternative from both the ideal and negative-ideal solutions [PKD23]. An alternative is deemed more preferable if it is geometrically closer to the ideal solution and farther from the negative-ideal solution. This approach provides a logical and computationally straightforward framework for resolving complex matching problems in capacity management and other logistical domains. In this study, we use TOPSIS framework as the core mathematical engine for our agent's matching policy, which is detailed in Section 5.3.3.

2.2 Reinforcement Learning

Reinforcement Learning (RL) encompasses a collection of methodologies for learning effective action policies in sequential decision-making problems [SB18]. The primary objective of reinforcement learning is to derive an optimal policy for an agent that maximizes the cumulative long-term reward. Sequential decision problems, which are integral to numerous domains involving uncertainty, appear in a variety of forms[Bis06]. Various methodological communities are actively engaged in formulating solutions for such problems. Prominent methodologies in this domain include robust optimization, optimal control, and simulation optimization[Pow22]. In a sequence of time steps in a process where decisions are made sequentially, each time step is denoted as "Period 1," "Period 2," and so forth, up to "Period N," indicating a temporal progression. At each period, a decision is made based on the available information, after which the process advances to the subsequent period, where the cycle is repeated until the final period "N" is reached.

2.2.1 Fundamentals of Reinforcement Learning

This section presents the fundamental principles of reinforcement learning, adopting the notation from [Put14]. Reinforcement learning enables an agent to learn optimal behaviors by directly interacting with its environment and observing the consequences of its actions. A Markov Decision Process (MDP) offers a structured framework for addressing learning through such interactions.

In this paradigm, an 'agent' is the entity that acquires knowledge and selects actions, while the 'environment' comprises everything external to the agent with which it interacts. As the agent executes actions, the environment responds to these choices, thereby presenting the agent with new circumstances to navigate.

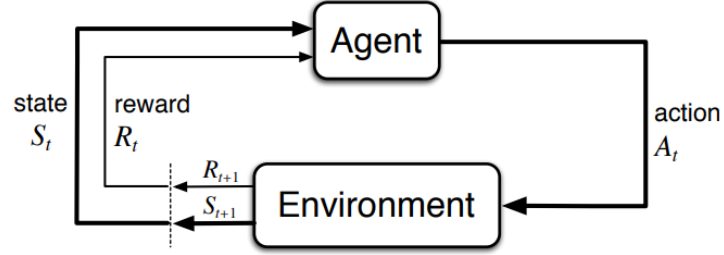


Figure 2.1: The agent-environment interaction in a Markov decision process.

An MDP is formally defined as a 4-tuple, (S, A, T, R) , where S represents a finite set of states, A is a finite set of actions, T is a transition function mapping from $S \times A \times S$ to probabilities, and R is a reward function that maps from S to real-valued rewards. A standard MDP is illustrated in Figure 2.1.

2.2.2 Fundamental Concepts and Notations

The interaction between an agent and its environment can be modeled as a Markov Decision Process (MDP), wherein the subsequent state is determined probabilistically from the preceding state and the action taken. The agent's objective is to learn an optimal policy that prescribes the action to be taken in each state. In many practical scenarios, the agent cannot directly observe the current state of the MDP but instead perceives features generated from the true state and the previous action, governed by a certain probability distribution. The underlying model in such cases is a Partially Observable MDP (POMDP) [SB18]. Table 2.1 provides a concise overview of the notations pertaining to MDPs and other relevant concepts used throughout this thesis [MRN05, Put14].

Table 2.1: Notations

Notation	Definition
A	Set of actions
S	Set of states
A_s	Set of actions available in state s
T	A mapping from $S \times A \times S$ to probability distributions over the states in S
R	A reward function that maps from S to real-valued rewards
Π	Set of policies $\{\pi_1, \pi_2, \dots, \pi_N\}$
S_t	State at time t
A_t	Action at time t
R_t	Reward at time t
γ	Discount rate (where $0 \leq \gamma \leq 1$)
I	Initial state of the MDP
π	A policy (a mapping from S to actions in A)
δ	A notation representing a policy
V, V^π	Value function under policy π , mapping elements of S to real values
π_t	Policy at time t in a non-stationary environment
V^{π_t}	Value function under the time-dependent policy π_t
$v_N^\pi(s)$	Value of the value function with policy π for state s in decision period N
$r_t(s, a)$	Reward for choosing action $a \in A_s$ in state s at decision period t
$p(\hat{s}, r \mid s, a)$	Probability of transitioning to state $\hat{s}$ and receiving reward r , given current state s and action a
$r_t(s, a, j)$	Reward received in period t when action $a \in A_s$ is chosen in $s \in S$ and the next state is j
$P_\delta(i, j)$	Transition probability under policy δ , given by $P_\delta(i, j) = T(i, \delta(i), j)$ for $i, j \in S$
$Q(s, a)$	State-action value function, representing the expected return from state s , taking action a , and following policy π thereafter
$Q^*(s, a)$	Optimal state-action value function, representing the maximum expected return from state s and action a under the optimal policy π^*
ϵ -optimal policy	A policy π that achieves a value within ϵ of the optimal value V^* for all states, i.e., $V^\pi(s) \geq V^*(s) - \epsilon$ for all $s \in S$
$M(o, i, a)$	Probability of observing o in state i after taking action a
$b_t(s)$	The belief-state; a probability distribution over states S at time t , where $b_t(s) = P[S_t = s]$
h	History of interactions up to the current time, including the sequence of past actions and observations
$\acute{o}, \acute{r}$	Observation and reward values at the next time step in a history-based decision process

2.2.3 Reinforcement Learning Pipeline

The canonical RL pipeline constitutes the foundational framework upon which all contemporary variations are constructed, with its mathematical underpinnings in the theory of MDPs [SB18, Bel57]. This pipeline is a continuous, iterative loop involving an agent and an environment, where the agent's actions influence its future observations and rewards.

Environment and Agent. The pipeline commences with the formalization of the problem domain as an *environment*. At any time step t , the environment is characterized by its state, $S_t \in S$. A state must adhere to the Markov property, which is expressed as $P(S_{t+1} \mid S_t, A_t, S_{t-1}, A_{t-1}, \dots) = P(S_{t+1} \mid S_t, A_t)$. The *agent* is the learning entity that observes S_t and selects an action, $A_t \in A(S_t)$. The agent's objective is to learn an optimal policy for selecting actions through a process of trial and error.

Reward Function and Objective. Upon the agent executing action A_t in state S_t , the environment transitions to a new state S_{t+1} and provides a scalar reward, R_{t+1} . The reward function, R , defines this mapping and the goal is specified in terms of reward function. The agent's objective is to maximize the expected cumulative discounted reward, or *return*, defined as $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$, where $\gamma \in [0, 1]$ is the discount factor. The conceptualization of all objectives as the maximization of this scalar return is known as the *reward hypothesis* [SB18].

Policy and Value Functions. The agent's behavior is dictated by its *policy*, π , which is a mapping from states to a probability distribution over actions. a probability distribution over actions. We will write $\pi(a \mid s) = P(A_t = a \mid S_t = s)$ to denote the probability of selecting a in state s . The goal is to determine an optimal policy, π^* , that maximizes the expected return from all states. To this end, RL algorithms learn a *value function* to estimate the long-term desirability of states or state-action pairs. The state-value function, $V^\pi(s)$, is the expected return starting from state s and following policy π . The action-value function, $Q^\pi(s, a)$, is the expected return after taking action a in state s and subsequently following policy π . These functions conform to the Bellman equations, which provide the theoretical basis for most RL algorithms.

The Learning Loop. The pipeline operates through *generalized policy iteration* (GPI), an iterative process in which the agent alternates between evaluating its policy (i.e., estimating the value function) and improving it [SB18]. This process necessitates managing the *exploration-exploitation trade-off*, which in-

volves balancing the utilization of current knowledge to accumulate rewards (exploitation) with the exploration of new actions to enhance its knowledge.

A **trajectory**, τ , is a sequence of states, actions, and rewards:

$$\tau = (S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, \dots)$$

The **return**, G_t , is the sum of discounted rewards from time-step t onward:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

where γ is a discount factor ($0 \leq \gamma \leq 1$) that balances the importance of immediate versus future rewards.

The foundational functions of reinforcement learning (RL) comprise a variety of mathematical and algorithmic constructs that are essential for defining and analyzing the RL framework. Table 2.2 provides an overview of these critical functions.

Table 2.2: Fundamental definitions in Reinforcement Learning

Notation	Definition
Deterministic policy π	$\pi(s) \in A_s$ for all $s \in S$.
Stochastic policy π	$\pi(a s) = P(A_t = a S_t = s)$ for all $s \in S$ and $a \in A_s$.
State-value function for policy π	$v_\pi(s) = \mathbb{E}_\pi[G_t S_t = s]$ for all $s \in S$.
Action-value function for policy π	$q_\pi(s, a) = \mathbb{E}_\pi[G_t S_t = s, A_t = a]$ for all $s \in S$ and $a \in A_s$.
Optimal state-value function v_*	$v_*(s) = \max_\pi v_\pi(s)$ for all $s \in S$.
Optimal action-value function q_*	$q_*(s, a) = \max_\pi q_\pi(s, a)$ for all $s \in S$ and $a \in A_s$.

Note on Expectation: The operator $\mathbb{E}_\pi[\cdot]$ denotes the **expected value** of a random variable (like the return G_t), given that the agent follows policy π . This means that all future actions are selected according to the action probabilities defined by $\pi(a|s)$.

2.2.4 Q-function

The **Q-function** (or action-value function) is central to many RL algorithms. **Q-learning** is a popular model-free algorithm that directly learns the optimal action-value function, $q_*(s, a)$, for an environment. This allows an agent to determine the best action to take in any state without needing a model of the environment's transition probabilities or reward function.

The core of the Q-learning algorithm is its update rule. After taking action A_t in state S_t and observing the immediate reward R_{t+1} and the next state S_{t+1} , the agent updates its estimate of the Q-value using the following formula:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

Where:

- $Q(S_t, A_t)$ is the current estimate of the action-value for the state-action pair.
- α is the **learning rate** ($0 < \alpha \leq 1$), which determines how much the new information overrides the old estimate.
- R_{t+1} is the reward received after taking action A_t .
- γ is the **discount factor** ($0 \leq \gamma < 1$).
- $\max_a Q(S_{t+1}, a)$ is the agent's estimate of the optimal future value it can get from its new state, S_{t+1} . This is the maximum Q-value over all possible actions a in the next state.

This update rule is a form of **Temporal-Difference (TD) learning**. It updates the value of the current state-action pair based on the observed reward plus the estimated value of the best action in the next state. By repeatedly applying this update, the function $Q(S, A)$ converges to the optimal action-value function, $q_*(s, a)$.

Strongly Connected Markov Chain. A Markov chain is strongly connected if there exists a directed path from each vertex to every other vertex.

Ergodic Markov Chain. A Markov chain is ergodic if it is possible to transition from any state to any other state.

Stationary and Nonstationary Policies. A non-stationary MDP is specified by the tuple (S, A, T, r, p) . The set S is a finite set of states. The collection

$A = \{A_s\}_{s \in S}$ contains a finite action set A_s for each state $s \in S$. We say that (s, a) is a state-action pair if $s \in S$ and $a \in A_s$. We denote $S = |S|$ and $A = (\sum_{s \in S} |A_s|)/S$. We denote T as the total number of time steps, and $r = \{r_t\}_{t=1}^T$ as the sequence of mean rewards.

Exogeneity. The quantities r_t and p_t vary across different t in general.

Endogeneity. In a non-stationary MDP instance (S, A, T, r, p) , given state s_t and action a_t , the system stochastically transitions to another state s_{t+1} , which is distributed as $p_t(\cdot | s_t, a_t)$, and receives a stochastic reward $R_t(s_t, a_t)$, which is 1-sub-Gaussian with mean $r_t(s_t, a_t)$ [SB18] [Put14].

Optimal Policy. The theory and algorithms of Markov decision processes for finite-horizon models are primarily concerned with identifying a policy π^* that maximizes the expected total reward and with defining this value. In other words, the objective is to find a policy π^* such that:

$$v_N^{\pi^*}(s) \geq v_N^{\pi}(s), \quad s \in S$$

Such a policy is termed an optimal policy. However, in certain models, an optimal policy may not be attainable. Therefore, the aim is to find an ε -optimal policy, that is, for an $\varepsilon > 0$, a policy π_ε^* with the property that:

$$v_N^{\pi_\varepsilon^*}(s) + \varepsilon > v_N^{\pi}(s), \quad s \in S$$

2.2.5 Partially Observable Environments

Consider a MDP (S, A, T, R) with finite sets S and A . Suppose that at time instant t , the Markov chain is in state $\varphi_t = i$, with $i \in S$. An agent must select an action a from the set of possible actions A and transition to a subsequent state j according to the transition probabilities $T(i, a, j)$. Observations are represented by the random variable O_t , which takes values in a finite set O . The random variables O_t are governed by the conditional probabilities:

$$P[O_t = o | \varphi_t = i, a_{t-1} = a]$$

Define the observation probability function M as:

$$M : O \times S \times A \rightarrow [0, 1]$$

$$M(o, i, a) = P[O_t = o | \varphi_t = i, a_{t-1} = a]$$

A POMDP is, therefore, a six-tuple (S, A, O, T, M, R) , where S, A , and O are the state-space, the action-space, and the observation-space, respectively, and T, M , and R are the transition probability function, the observation probability function, and the reward structure, respectively. In a POMDP, the agent is generally unable to observe the underlying state φ_t of the Markov chain. As such, to choose an action based on the observations it receives, the agent must maintain an estimate of the current state of the Markov chain. This estimate is referred to as the belief-state or information-state and possesses important properties that will be explored subsequently.

2.2.6 The Belief-State

In a POMDP, the agent's estimate of the current state is a probability distribution known as the belief-state. We will denote the belief at time t as b_t , where $b_t(s) = P[S_t = s]$ is the probability that the system is in state $s \in S$.

Assume the belief at time t is $b_t(s)$. If the agent chooses action $A_t = a$, the system will transition to state s' with probability $T(s, a, s')$. The probability of the next state being s' given the action a is the sum over the probabilities of all possible current states:

$$\begin{aligned} P[S_{t+1} = s' | A_t = a] &= \sum_{s \in S} P[S_{t+1} = s' | S_t = s, A_t = a] P[S_t = s] \\ &= \sum_{s \in S} T(s, a, s') b_t(s) \end{aligned}$$

Given that the agent chose action a , observed o , and had a prior belief $b_t(s)$, the new belief over the next state $S_{t+1} = s'$, denoted $b_{t+1}(s')$, is given by Bayes' rule:

$$b_{t+1}(s') = \frac{\sum_{s \in S} T(s, a, s') M(o, s', a) b_t(s)}{\sum_{s'' \in S} \sum_{s \in S} T(s, a, s'') M(o, s'', a) b_t(s)}, \quad s' \in S$$

2.2.7 DDPG

The Deep Deterministic Policy Gradient (DDPG) algorithm is a model-free, off-policy actor-critic method that utilizes deep function approximators to learn policies in high-dimensional, continuous action spaces. It amalgamates concepts from Deterministic Policy Gradient (DPG) and Deep Q-Networks (DQN), addressing function approximation error in actor-critic methods by employing a

deep neural network to learn a deterministic policy and a Q-function. DDPG introduces target networks to stabilize training and uses a replay buffer to decorrelate consecutive samples, thereby enhancing learning efficiency and stability [SLH⁺14].

In essence, it employs two target networks, analogous to the Actor-Critic method:

1. **Actor:** Proposes an action given a state.
2. **Critic:** Predicts the quality of the action, given a state and an action.

The pseudocode for the DDPG is presented in algorithm 1.

Algorithm 1 DDPG algorithm

- 1: Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ .
 - 2: Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$.
 - 3: Initialize replay buffer R .
 - 4: **for** episode = 1, M **do**
 - 5: Initialize a random process $\mathcal{N}$ for action exploration.
 - 6: Receive initial observation state s_1 .
 - 7: **for** $t = 1, T$ **do**
 - 8: Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise.
 - 9: Execute action a_t and observe reward r_t and new state s_{t+1} .
 - 10: Store transition (s_t, a_t, r_t, s_{t+1}) in R .
 - 11: Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R .
 - 12: Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$.
 - 13: Update critic by minimizing the loss: $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$.
 - 14: Update the actor policy using the sampled policy gradient:
 - 15: $\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$.
 - 16: Update the target networks:
 - 17: $\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}$
 - 18: $\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$
 - 19: **end for**
 - 20: **end for**
-

2.2.8 Automated Reinforcement Learning

Reinforcement learning has demonstrated its capacity to identify optimal solutions through interaction with the environment in a variety of applications. When integrated with neural networks, significant achievements in complex domains such as board games, as documented in [SHM⁺16], have highlighted

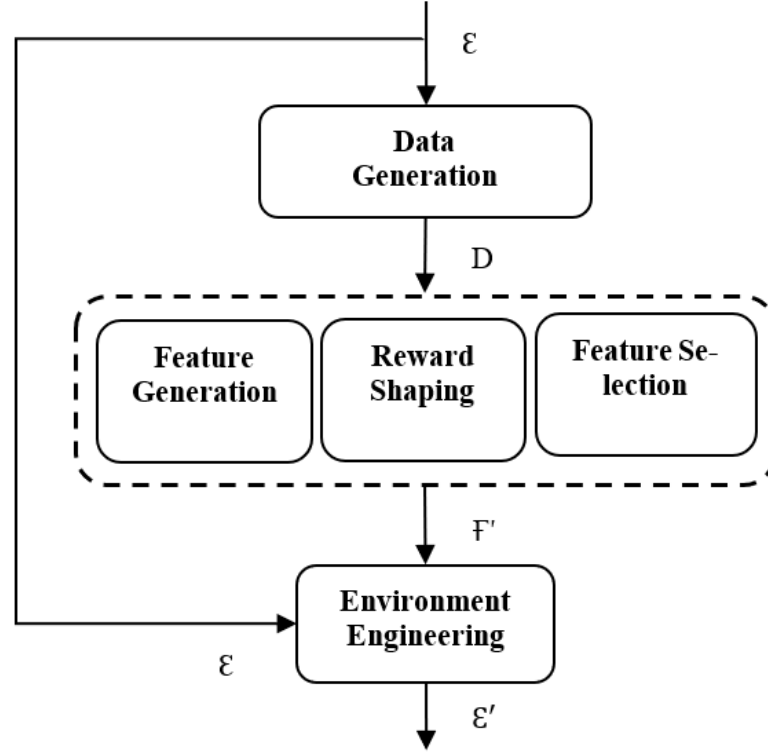


Figure 2.2: Feature Engineering stage in the Online Pipeline.

the potential of this approach, stimulating researchers to develop a wide array of RL algorithms tailored to specific problem characteristics.

In [PHRS⁺22], a general Automated Reinforcement Learning(AutoRL) pipeline for solving problems is introduced. The primary research questions in AutoRL concern the optimization of hyperparameters with minimal computational resources and the modeling of a problem as an MDP, as well as generalizing the mapping from available information to an RL environment. In [MLM⁺23], pipelines for both offline and online RL are proposed:

- **Online Pipeline.** The Online AutoRL Pipeline takes an environment ξ as input, which is modified by a Feature Engineering stage to facilitate learning in the subsequent stage. Using the features created in this stage, it outputs a transformed environment ξ' . Through the Policy Generation process, the environment ξ' is used to estimate the optimal policy $\hat{\pi}^*$. In the Policy Evaluation phase, a performance estimate, $\eta(\hat{\pi}^*)$, is provided based on a performance index η . Feature engineering, policy generation, and policy evaluation are the three main components of the Online AutoRL pipeline.

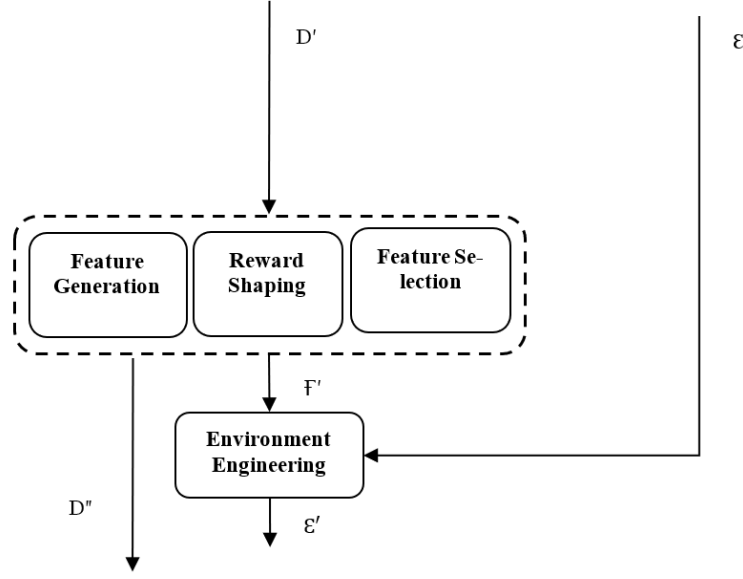


Figure 2.3: Feature Engineering stage in the Offline Pipeline.

- Offline Pipeline.** The Offline AutoRL Pipeline, in contrast to the online version, includes two additional preliminary stages: Data Generation and Data Preparation. When an environment ξ is provided as input, the Data Generation stage creates a dataset D . If a dataset D is already available, for instance from a real-world process, this step is omitted. Consequently, the environment ξ is only used to evaluate the performance of a policy. By applying corrections to each row of the dataset D , the Data Preparation stage modifies the dataset D to obtain D' . After passing through the Feature Engineering stage, the environment ξ and dataset D' are transformed into a dataset D'' and an environment ξ' equipped with transformed states, actions, and rewards. In the Policy Generation stage, the dataset D'' is used to estimate the optimal policy $\hat{\pi}^*$. This differs from the online stage in that it uses the dataset D'' , whereas the Policy Evaluation stage uses the environment ξ' for estimating $\eta(\hat{\pi}^*)$. The Offline AutoRL Pipeline is illustrated in Figure 2.3.

2.3 Reward Machines

While the canonical Reinforcement Learning (RL) framework provides a robust foundation for sequential decision-making, its practical application to complex, real-world problems often confronts significant challenges related to sample inefficiency [IKVM22]. Many sophisticated RL algorithms treat the reward function, R , as an opaque or "black-box" signal. This necessitates extensive

exploration for an agent to discern which sequences of actions lead to high cumulative returns. This limitation becomes particularly pronounced in environments characterized by sparse or delayed rewards, where the feedback is so infrequent that the credit assignment problem becomes exceptionally difficult [KT23]. Furthermore, the manual design of reward functions to guide an agent towards desired behaviors without introducing unforeseen loopholes—a process known as reward engineering—is a notoriously arduous task [TIKVM18].

In response to these limitations, a considerable body of research has explored the use of automata-based models to represent task structures and reward functions more explicitly. These formalisms provide a structured and unambiguous medium for specifying complex, temporally extended objectives that are difficult to express with simple Markovian rewards [Cea19]. Among these, the **Reward Machine (RM)** has emerged as a particularly influential framework [TIKVM18, IKVM22]. An RM is a type of finite-state automaton that encodes the underlying structure of a task's reward function. By representing the task as a graph of states and transitions that are triggered by high-level environmental events, an RM makes the logic of the reward function transparent to the learning agent [LCK⁺22]. This "white-box" approach fundamentally alters the learning problem and provides the basis for a suite of algorithms designed to accelerate policy acquisition.

2.3.1 Integration with MDPs

The conceptual core of the Reward Machine framework is its formal definition as a finite-state automaton that dictates the issuance of rewards based on task progression. This structure provides a formal basis for decomposing complex problems into more manageable components.

Formally, a Reward Machine is a tuple $\mathcal{R} = \langle U, u_I, F, AP, \delta_u, \delta_r \rangle$, where:

- U is a finite set of machine states.
- $u_I \in U$ is the initial state of the machine.
- $F \subseteq U$ is a set of terminal machine states.
- AP is a finite set of *atomic propositions*, which represent high-level, human-interpretable events that can occur in the environment (e.g., 'key_picked_up', 'door_opened').
- $\delta_u : U \times 2^{AP} \rightarrow U \cup F$ is the state-transition function. It maps a current

machine state and a set of true atomic propositions to a subsequent machine state.

- $\delta_r : U \times 2^{AP} \rightarrow \mathbb{R}$ is the reward-issuing function, which outputs a scalar reward based on the current machine state and the set of true atomic propositions.

The critical link between the agent’s low-level experience within the environment’s MDP and the high-level logic of the RM is established through a **labelling function**, denoted as L . This function, $L : S \times A \times S \rightarrow 2^{AP}$, abstracts a low-level environmental transition from state S_t to S_{t+1} via action A_t into a set of true atomic propositions from AP [IKVM22]. This set of propositions then drives the state transitions and reward emissions of the RM.

When an RM is composed with a standard MDP, the result is a **product MDP**, sometimes referred to as an MDPRM. In this new process, the state space is the Cartesian product of the environment states and the RM states, $S \times U$. An agent’s policy, π , is subsequently learned over this augmented state space, becoming $\pi(a_t | s_t, u_t)$, where $s_t \in S$ is the environment state and $u_t \in U$ is the RM state at time t . This composition is of great theoretical importance, as it can transform a problem with a non-Markovian reward function (where the reward depends on the history of states) into a fully Markovian decision process over the augmented state space (s, u) . The RM state u_t effectively serves as a compact memory of all task-relevant history, satisfying the Markov property for the augmented state.

2.3.2 Foundational Reward Machine Algorithms

The transparent structure of a given RM enables the development of several powerful algorithmic techniques that can significantly improve learning efficiency compared to standard, black-box RL approaches.

One of the first and most direct methods is **Q-Learning for Reward Machines (QRM)** [TIKVM18]. This algorithm decomposes the primary learning task by training a separate action-value function, $Q_u(s, a)$, for each state $u \in U$ of the reward machine. Since Q-learning is an off-policy algorithm, a single stream of environmental interaction $(S_t, A_t, R_{t+1}, S_{t+1})$ can be used to provide updates for all of these distinct Q-functions in parallel. This parallel learning process leads to substantial improvements in sample efficiency.

A second technique involves the use of **automated potential-based reward**

shaping. The graphical structure of an RM allows for the automatic derivation of a valid potential function, $\Phi(u)$, typically by performing value iteration on the RM states themselves [IKVM22]. This potential function provides a dense, intermediate reward signal, $\gamma\Phi(u_{t+1}) - \Phi(u_t)$, which can guide the agent's exploration towards achieving task subgoals. Crucially, when constructed correctly from the RM, this shaping reward is guaranteed to be policy-invariant, meaning it does not alter the optimal policy of the underlying problem.

Arguably the most potent foundational method is **Counterfactual Reasoning for Reward Machines (CRM)** [IKVM22]. This advanced off-policy approach allows an agent to generate multiple synthetic, or "counterfactual," learning experiences from a single environmental transition. For a single observed transition (s_t, a_t, s_{t+1}) , the agent can reason about what would have happened if it had been in a different RM state $u' \neq u_t$. This enables the agent to update the Q-function $Q_{u'}(s_t, a_t)$ for every RM state u' simultaneously. This technique dramatically enhances data reuse and further accelerates the speed of learning. These foundational methods highlight the value of exploiting the explicit task structure provided by an RM, but they operate on the assumption that such an RM is provided beforehand.

2.4 Real-World Challenges in Reinforcement Learning

Despite the remarkable successes of Reinforcement Learning (RL) in simulated environments, the transition to real-world systems is impeded by numerous practical difficulties. Many seminal RL achievements result from highly tailored implementations that do not generalize well to other problems. To effectively deploy RL in practical applications, a series of significant challenges must be systematically addressed. As catalogued by Dulac-Arnold et al. [DALM⁺21], these core challenges can be summarized as follows:

1. Learning from a limited number of samples.
2. Managing substantial delays in observations, actions, or rewards.
3. Operating within high-dimensional state and action spaces.
4. Adhering to critical system and safety constraints.
5. Interacting with systems that are partially observable or non-stationary.

6. Learning from multi-objective or inadequately specified reward functions.
7. Executing policy decisions swiftly to meet real-time demands.
8. Training policies offline using fixed, pre-existing datasets.
9. Generating policies that are explainable to human operators.

These multifaceted difficulties can be organized into a structured taxonomy, which facilitates a clearer analysis of the obstacles to real-world RL deployment. We adopt a classification that groups these nine challenges into three superordinate categories:

- **Data-Related Challenges:** Difficulties pertaining to the availability, cost, and intrinsic nature of the data utilized for learning (Challenges 1 and 8).
- **Environment-Related Challenges:** Complexities inherent to the system’s dynamics, observations, objectives, and operational constraints (Challenges 2, 3, 4, 5, and 6).
- **Operational Challenges:** Practical and logistical hurdles associated with the deployment and live interaction of an RL agent within a production system (Challenges 7 and 9).

The subsequent sections delve into each of these categories, elucidating the specific problems within them and surveying prominent research efforts aimed at their resolution.

2.4.1 Data-Related Challenges

Limited Sample Efficiency

A primary obstacle in real-world RL is the necessity to learn from a restricted volume of interactions. In contrast to simulations where data generation is inexpensive, acquiring samples from physical or production systems is often costly, time-consuming, or carries inherent risks. This reality demands algorithms with high sample efficiency. A range of methodologies has been developed to address this issue. Model-based RL, for example, seeks to learn a dynamics model of the environment, which can then be used for planning or to generate simulated experiences, thereby reducing the required number of real-world interactions [CCML18]. Another prominent approach is meta-learning, where an agent is pre-trained across a distribution of related tasks, enabling it to adapt to a novel task with minimal data [FAL17]. Furthermore, learning can be significantly

accelerated by leveraging expert demonstrations to bootstrap the policy, a technique effectively implemented in Deep Q-learning from Demonstrations (DQfD) [HVP⁺18].

Offline Reinforcement Learning

Many real-world systems cannot permit online interaction for learning due to safety or operational constraints. However, these systems often possess extensive logs of historical data from prior operations. The field of offline RL, or batch RL, focuses on deriving optimal policies exclusively from such static datasets. A principal difficulty in this paradigm is the distributional shift between the behavior policy that generated the data and the new policy being learned, which can cause severe errors for standard off-policy algorithms [LKTF20]. To counteract this, conservative methods have been proposed, which penalize deviations from the actions present in the dataset. Notable examples include Conservative Q-Learning (CQL) [KL19] and Batch-Constrained Q-learning (BCQ) [FMP19], both of which regularize the learning process to avoid actions outside the data support.

2.4.2 Environment-Related Challenges

System Delays

Real-world systems frequently exhibit considerable delays in actuation, sensing, or reward feedback, violating the standard Markov Decision Process (MDP) assumption of instantaneous state transitions. These latencies introduce a form of partial observability and complicate the credit assignment problem. A common strategy to mitigate this is to augment the state representation with a history of recent actions and observations, allowing the policy to implicitly learn and compensate for the delays [HS13]. For tasks characterized by substantial reward delays, specialized algorithms are necessary. The RUDDER algorithm, for instance, employs a learned model to redistribute delayed rewards to the causally responsible preceding actions, facilitating more effective learning [AMGW⁺19].

High-Dimensional State and Action Spaces

Many practical applications, from controlling multi-jointed robots to managing large-scale recommender systems, involve state and action spaces of extremely high dimensionality. This "curse of dimensionality" poses significant challenges for exploration and generalization. Deep learning has been instrumental in this domain, using neural networks as powerful function approximators for policies

and value functions [MKS⁺15]. To manage vast action spaces, some methods structure the decision-making process, for instance, by employing a two-step approach of candidate generation followed by ranking [CAS16], or by using nearest-neighbor techniques to efficiently search for viable actions in a learned embedding space [DAEvH⁺15].

Satisfying Environmental Constraints

An agent operating in a real-world system must adhere to strict operational constraints to ensure safety and prevent damage. This requirement has given rise to the field of Safe RL, which is often formalized using the Constrained Markov Decision Process (CMDP) framework [Alt04]. In a CMDP, the agent’s objective is to maximize a reward function while ensuring that the expected value of one or more cost functions remains below a specified threshold. Algorithms like Constrained Policy Optimization (CPO) extend standard policy gradient methods to incorporate these safety constraints directly into the optimization process [AHTA17]. Another approach involves the use of a safety layer or shield, which monitors the agent’s proposed actions and intervenes by projecting any unsafe action onto the nearest safe alternative [DDV⁺18].

Partial Observability and Non-Stationarity

An agent in a real-world setting rarely perceives the complete state of the environment; it must operate based on partial and often noisy observations. This condition is modeled by the Partially Observable Markov Decision Process (POMDP). Furthermore, the dynamics of the environment may change over time—a phenomenon known as non-stationarity. The standard approach to handling partial observability is to equip the agent with memory, commonly by integrating Recurrent Neural Networks (RNNs) into the policy or value network architecture [HS15]. To cope with non-stationarity, adaptive algorithms are employed. These methods may involve continually updating a model of the system’s dynamics and re-planning accordingly [FAL17], or using techniques like domain randomization during training to expose the policy to a wide range of possible dynamics [PAZA18].

Multi-Objective Reward Functions

The objectives for a real-world system are seldom captured by a single scalar reward. More typically, performance is a trade-off between multiple, often conflicting, goals (e.g., maximizing production while minimizing energy consumption). This is the domain of multi-objective reinforcement learning (MORL)

[RVWD13]. MORL algorithms aim to find a set of Pareto-optimal policies that represent different optimal trade-offs. Some methods achieve this by learning a separate value function for each objective [VSFR⁺17], while others learn a single, preference-conditioned policy that can generate diverse solutions without retraining.

2.4.3 Operational Challenges

Real-Time Inference

For many applications, particularly those involving interaction with physical systems or users, decisions must be computed within a strict time budget. A computationally intensive policy is impractical if its inference latency exceeds the system’s control frequency. This operational constraint necessitates the use of lightweight policy representations and efficient algorithms. A common technique is policy distillation, where a large, high-performance policy is compressed into a smaller, faster network that approximates its behavior [RCG⁺15]. Alternatively, algorithms can be designed explicitly for real-time performance, for example, by using computationally efficient models like decision trees for rapid planning [HS13].

2.4.4 Real-World Reinforcement Learning benchmark

[DALM⁺21] introduced the *realworldrl-suite*, a benchmark for reinforcement learning. Within this suite, hyperparameters are configured to generate three distinct classes of environmental complexity: easy, medium, and hard. As noted in their research, algorithms such as DDPG exhibit poor performance on the ‘hard’ classification of this benchmark. The introduction of such a benchmark is a valuable contribution to the field of reinforcement learning, as it underscores the necessity of evaluating algorithms across varying levels of complexity to ensure robust and generalizable results. Table 2.3 outlines the specific RL environment settings for each challenge.

Table 2.3: The RL environment settings for each challenge[DALM⁺21]

Experiment	Challenge 1 (easy)		Challenge 2 (medium)		Challenge 3 (hard)	
System delays	Time steps		Time steps		Time steps	
Action	3		6		9	
Observation	3		6		9	
Rewards	10		20		40	
Action repetition	1		2		3	
Gaussian noise	SD		SD		SD	
Action	0.1		0.3		1.0	
Observation	0.1		0.3		1.0	
Stuck/dropped noise	Prob.	Time steps	Prob.	Time steps	Prob.	Time steps
Stuck sensor	0.01	1	0.05	5	0.1	10
Dropped sensor	0.01	1	0.05	5	0.1	10
Perturbation	[Min,Max]	Std.	[Min,Max]	Std.	[Min,Max]	Std.
Cartpole	[0.9,1.1]	0.02	[0.7,1.7]	0.1	[0.5,2.3]	0.15
quadruped	[0.25,0.3]	0.005	[0.25,0.8]	0.05	[0.25,1.4]	0.1
Walker	[0.225,0.25]	0.002	[0.225,0.4]	0.015	[0.15,0.55]	0.04
Humanoid	[0.6,0.8]	0.02	[0.5,0.9]	0.04	[0.4, 1.0]	0.06
High dimensionality	State dimension		State dimension		State dimension	
Increase	10		20		50	

2.5 Extending Real-World Reinforcement Learning Benchmarks

In the preceding section, we discussed various challenges associated with real-world reinforcement learning. Furthermore, we presented a summary table delineating key hyperparameters defined by previous research to categorize complexity classes within the RealWorldRL suite. While the existing benchmark primarily addresses fundamental robotic tasks, this section aims to extend the evaluation to two additional challenging scenarios. First, we introduce Reconnaissance Blind Chess, a partially observable chess environment designed to test decision-making under uncertainty. Subsequently, we highlight the "Challenge of Hyperparameter Optimization" through an exploration of the real-world "Footstep Biometrics" dataset, emphasizing the difficulties associated with selecting optimal hyperparameters for effective model training.

2.5.1 A Quantitative Framework for Evaluating Deep Learning Training Pipelines

The efficacy of a deep learning model is profoundly influenced by a multitude of design choices and configuration parameters that must be established prior to the training phase. This process of hyperparameter optimization is a critical and challenging aspect of developing robust machine learning systems [FID22]. The selection of an optimal configuration from a vast search space necessitates a systematic approach to experimentation and evaluation. The primary challenge lies in defining a comprehensive set of hyperparameters that govern both the architectural design of the network and the dynamics of the training algorithm itself. Key parameters that require careful consideration include:

- **Optimizer:** The choice of optimization algorithm dictates the strategy by which the model's parameters (weights) are iteratively adjusted to minimize a given loss function. Algorithms such as Stochastic Gradient Descent (SGD) and its variants, or adaptive methods like Adam, employ different mechanisms for navigating the loss landscape, which can significantly impact convergence speed and final model performance [KB14] [Rud16].
- **Scheduler:** A learning rate scheduler provides a framework for dynamically adjusting the learning rate during the training process. A common

strategy involves starting with a relatively high learning rate to facilitate rapid initial progress and gradually decreasing it to allow for fine-grained parameter adjustments as the model approaches a minimum in the loss function. This can lead to faster convergence and improved generalization [GBCB16] [PGM⁺19] .

- **Layer Sizes:** The architecture of a neural network is defined by its depth (the number of layers) and width (the number of neurons within each layer). These parameters collectively determine the model's capacity, which is its ability to approximate complex functions. A model with insufficient capacity may underfit the data, while one with excessive capacity is prone to overfitting [APH⁺21].
- **UMAP Distance:** When employing dimensionality reduction techniques such as Uniform Manifold Approximation and Projection (UMAP) for data preprocessing or feature extraction, its own hyperparameters become part of the pipeline's configuration. The `min_dist` parameter, for instance, controls the minimum distance between embedded points in the low-dimensional space, thereby influencing the visual tightness of the resulting clusters and the preservation of local versus global data structure [MHM18].
- **Number of Channels:** In the context of Convolutional Neural Networks (CNNs), the number of channels in a convolutional layer corresponds to the number of filters applied to the input volume. Each filter is trained to detect specific features, and thus, the number of channels (or filters) determines the richness of the feature representation that the layer can learn [GBCB16].
- **Horizontal Flip:** This is a common data augmentation technique wherein training images are randomly flipped along the vertical axis. This process artificially expands the training dataset and encourages the model to learn representations that are invariant to the horizontal orientation of objects, which can significantly improve generalization performance [SK19].
- **Learning Rate:** As a fundamental hyperparameter for gradient-based optimization, the learning rate controls the magnitude of the weight updates at each iteration. The selection of an appropriate learning rate is critical; a value that is too small can lead to excessively slow convergence, while a value that is too large can cause the optimization process to become

unstable or fail to converge entirely [GBCB16].

Given the extensive set of configurable options, a common methodology involves training multiple models, each with a distinct configuration, and subsequently comparing their performance. For each training run, a detailed log is generated, capturing the model’s learning dynamics on an epoch-by-epoch basis. These logs typically contain several key metrics that provide insight into the training process [FID22]:

- **Epoch:** An epoch signifies one complete forward and backward pass of all the training examples through the network. Deep learning models are trained for multiple epochs, allowing the parameters to be iteratively refined [GBCB16].
- **Loss:** This metric quantifies the model’s error by measuring the discrepancy between its predictions and the true target values. The primary objective of training is to minimize this value [GBCB16].
- **Accuracy:** For classification tasks, accuracy measures the proportion of predictions that the model correctly identifies. It provides a direct and interpretable measure of the model’s correctness [JS11].
- **Validation Loss & Validation Accuracy:** These metrics are identical to their training counterparts but are computed on a separate, held-out validation dataset. The model does not learn from this data; instead, these metrics serve as a crucial proxy for the model’s ability to generalize to new, unseen data. A significant divergence between training and validation performance is a hallmark of overfitting [GBCB16].

2.6 A Benchmark for Footstep Biometrics

Progress in biometric footstep recognition has been historically limited by the absence of large, diverse datasets required to develop and validate robust models. The recent introduction of the UNB StepUP-P150 dataset marks a significant milestone, providing an unprecedented resource of approximately 200,000 high-resolution footstep pressure recordings from 150 individuals across varied conditions. This dataset was the centerpiece of the First International StepUP Competition for Biometric Footstep Recognition, a challenge designed to accelerate progress towards real-world deployment [LPS⁺25].

The competition tasked participants with developing robust verification models

capable of overcoming two primary challenges: generalisation to new users for whom only limited reference data is available, and adaptation to novel conditions, including unseen footwear and walking speeds. There are three subsets in the competition dataset:

- **Training Set:** Comprised the full UNB StepUP-P150 dataset, containing over 200,000 footstep samples from 150 individuals. These samples were collected under four different footwear conditions and four distinct walking speeds, providing a rich and diverse foundation for model training.
- **Reference Set:** This set was designed to simulate a realistic user enrolment scenario. It consisted of 150 samples from 15 new individuals (not present in the training set), with each user providing ten footsteps (five left, five right) under a single condition: wearing their own shoes at a preferred walking speed.
- **Probe Set:** A dedicated test set of 10,000 footsteps was used for evaluation. These probes were drawn from 30 individuals, including the 15 enrolled users from the reference set and 15 entirely unseen users. The samples covered three footwear types and three walking speeds, with approximately 90% of the genuine user samples representing conditions not seen in the reference set. Each probe was associated with a claimed identity, with 30% being true claims and 70% being false claims.

Evaluation Protocols in Biometric Systems

The evaluation of biometric systems necessitates metrics that provide a balanced assessment of system performance. In line with previous major biometric competitions, such as the Fingerprint Verification Competitions (FVC) [MMC⁺04], the StepUP challenge primarily ranked solutions based on the Equal Error Rate (EER). EER is defined as the point on the detection error tradeoff curve where the False Match Rate (FMR) and the False Non-Match Rate (FNMR) are equal [MW02]. FMR is the proportion of impostor attempts that are incorrectly accepted, while FNMR is the proportion of genuine attempts that are incorrectly rejected. The EER is a valuable metric because it provides a single, threshold-independent measure of a system's ability to balance security (low FMR) and convenience (low FNMR).

The selection of EER as the primary metric was also motivated by the competition's design. Since competitors were not informed of the proportion of true and false claims in the probe set, it would have been difficult to tune a spe-

cific decision threshold accurately. EER circumvents this issue by evaluating the intrinsic separability of the learned feature space, irrespective of a chosen operating point. While EER was the primary metric, supplementary metrics including overall accuracy (ACC), balanced accuracy (BACC), and FMR100 (the FNMR at a fixed FMR of 1%) were also reported to offer a more comprehensive view of performance [LPS⁺25]. This evaluation methodology aligns with the terminology and guidelines established by the ISO/IEC 19795 standard for biometric performance testing [MW02].

2.7 The Omnigenomic Benchmark (OGB)

To facilitate a standardized and rigorous evaluation of genomic foundation models, the Omnigenomic Benchmark (OGB) was employed in this study. The OGB was developed to address the need for a more comprehensive assessment framework in genomics, a field where generative artificial intelligence is projected to unlock between \$60 and \$110 billion in annual economic value [YL25]. The benchmark is designed to assess models along three critical axes: generalization across diverse species, integration of multiple data modalities like sequence and structure, and the generation of fine-grained, nucleotide-level predictions [YL25]. The following sections, which detail the benchmark’s tasks and evaluation methodology, have been adapted from the official OGB documentation [YL25].

2.7.1 OGB Task Descriptions

The OGB is composed of a suite of in-silico tasks designed to test a model’s capabilities on a wide range of genomic problems. The primary tasks are described below [YL25] [Cha25].

- **mRNA Degradation Rate Prediction:** In this regression task, the model is required to predict nucleotide-level decay rates within mRNA sequences. This information is fundamental for understanding gene expression and designing stable RNA therapeutics [YL25].
- **Single-Nucleotide Mutation Detection (SNMD):** This is a token-level binary classification task. The objective is to identify the presence of single-nucleotide mutations in synthetic plant RNA, which is a key challenge in agricultural genomics [YL25].

- **Single-Nucleotide Mutation Repair (SNMR):** Building upon SNMD, this four-way classification task evaluates a model’s ability to not only detect a mutation but also to predict the correct base (A, U, C, or G) for the mutated site. This capability is relevant to precision medicine and gene therapy research [YL25].
- **RNA Secondary Structure Prediction (SSP):** This task assesses a model’s capacity to predict the base-pairing patterns that form an RNA molecule’s secondary structure. Accurate structure prediction is essential for functional RNA analysis. The task is evaluated on three separate datasets: **bpRNA**, **ArchiveII**, and **RNAStrAlign** [YL25].
- **Antibiotic Resistance Class Prediction (ARG-AP):** A sequence-level classification task where the goal is to predict which class of antibiotic a given gene provides resistance to. This task is aimed at supporting antimicrobial resistance research by enabling faster identification of resistance-conferring genes [YL25].

2.7.2 Evaluation and Scoring Framework

A unified framework was established by the OGB to ensure a fair and robust comparison of models across all tasks. The final ranking is determined by a single aggregate score derived from a multi-stage process [YL25] [Cha25].

1. Task-Specific Metrics Each task is first evaluated using a metric suited to its objective. For the regression-based mRNA Degradation task, Inverse Root Mean Squared Error (iRMSE) is used. For the binary classification SNMD task, the Area Under the ROC Curve (AUC-ROC) is employed. For all other multi-class classification tasks (SNMR, ARG-AP, and all three SSP datasets), the Macro-F1 score is calculated [YL25].

2. Score Normalization To enable aggregation, all task-specific metrics are normalized to a consistent 0–1 scale, where a higher value indicates better performance. For metrics where lower is better, such as RMSE, the official OGB documentation specifies the use of an inverse normalization formula [YL25]:

$$\text{Normalized iRMSE} = \frac{\max(\text{iRMSE}) - \text{iRMSE}}{\max(\text{iRMSE}) - \min(\text{iRMSE})} \quad (2.1)$$

3. Weighted Aggregation and Ranking A final score is computed as the weighted sum of all normalized task scores. The weights are assigned based

on the task’s importance within the benchmark. The weights for mRNA Degradation, SNMD, SNMR, and Antibiotic Resistance are each set to 1, while the three Secondary Structure Prediction tasks (bpRNA, ArchiveII, RNAStrAlign) are each assigned a weight of 2/3 [YL25]. The final score is calculated using the following formula:

$$\text{Final Score} = \sum_{i=1}^N w_i \cdot s_i \quad (2.2)$$

where N represents the total number of tasks, w_i is the weight for task i , and s_i is the normalized score for that task. Models are ultimately ranked based on this final aggregate score [YL25].

2.8 Reconnaissance Blind Chess

The study of strategic decision-making represents a cornerstone of inquiry across economics, computer science, and artificial intelligence. While initial triumphs in AI were frequently demonstrated in games of perfect information, where the entire game state is common knowledge, a more extensive and arguably more pertinent class of real-world problems involves interactions characterized by hidden, incomplete, or asymmetric information. Such scenarios are formalized as Partial Information Games (PIGs), presenting a fundamentally distinct and more intricate set of challenges. In a PIG, at least one participant possesses incomplete knowledge of the prevailing game state, S_t . This informational deficit necessitates that a player’s policy, π , be formulated for an *information set*—a collection of decision nodes that the player cannot distinguish from one another. Consequently, the strategic approach must be applicable across all states within the same information set, establishing the foundational complexity that distinguishes these games from their perfect-information counterparts.

Variations in Chess as a Research Platform Chess, with its history as a proving ground for artificial intelligence, provides an ideal environment for exploring complex decision-making paradigms. While its strategic depths have been extensively explored by high-performance engines in perfect-information contexts, its intrinsic complexity renders it an ideal environment for evaluating performance under novel and severe constraints. A notable departure from the prevailing trend of leveraging ever-increasing computational power was the FIDE & Google Efficient Chess AI Challenge, which deliberately shifted the focus from "brute-force computation to elegant and efficient design" [FID22]. This initiative, which tasked participants with creating "smart, resource-light

chess engines," resonates with Claude Shannon's foundational vision of the chess-playing computer as a "wedge" for addressing other complex problems of a similar nature [Sha50]. This reframing underscores a critical trajectory in AI research: the development of robust intelligence capable of operating effectively under the "fog and friction" of real-world limitations, whether they be computational, energetic, or, as in the case of Reconnaissance Blind Chess, informational.

Example: The FIDE & Google Efficient Chess AI Challenge To illustrate the nature of constrained AI competition, consider the standard chess environment. In this setting of perfect information, an agent possesses a complete observation of the entire board state at all times. An engine such as Stockfish [The25] can be employed as a formidable agent within this environment. Its proficiency can be modulated via a "Skill Level" parameter, ranging from 0 (weakest) to 20 (strongest). The results of matches between engines of varying skill levels demonstrate a clear hierarchy of performance, with higher skill levels consistently dominating lower ones. Given the complete observability of the board, a Stockfish agent operating at skill-level 20 can be regarded as the preeminent autonomous agent in this domain. The FIDE & Google Efficient Chess AI Challenge [Fé25] introduced a radical constraint by limiting the size of the agent's executable file to a mere 64 KB. Considering that the standard Stockfish engine is approximately 80 MB, the challenge for machine learning practitioners is to develop an agent that approaches expert-level performance while adhering to this severe size limitation.

Reconnaissance Blind Chess The conceptual foundation of Reconnaissance Blind Chess (RBC) is the partial observability of the environment. In essence, an agent is restricted to sensing a small section of the board at each turn and must make a decision based on this limited information. For instance, as illustrated in Figure 2.4, the White player may only sense the portion of the board demarcated by the red rectangle. White must then execute a move despite being oblivious to the remainder of the board, where, in this hypothetical scenario, a pawn was recently captured by a Black knight.

The history of artificial intelligence is inextricably linked with the study of games. From early investigations into checkers to landmark achievements in chess [CHJH02], Go [SHM⁺16, SHS⁺17], and poker [BS18, BS19], games have provided well-defined, reproducible environments for developing and testing algorithms for sequential decision-making. These successes, however, have

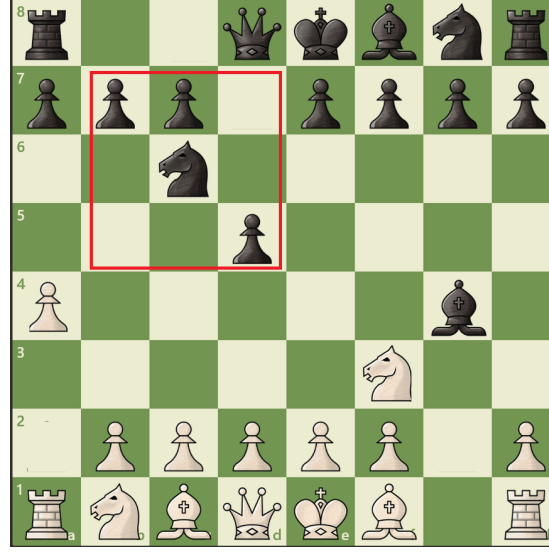


Figure 2.4: A visual representation of partial observability in RBC. The White player senses a 3x3 portion of the board (red rectangle) and must move without knowledge of events outside this view, such as a recent capture.

been largely concentrated in domains of perfect information or in games where substantial common knowledge guides strategic reasoning. A significant, less-explored frontier persists: decision-making in environments defined by vast private information, minimal common knowledge, and the necessity of strategic planning under profound uncertainty [GPS⁺22].

Reconnaissance Blind Chess was developed by researchers at the Johns Hopkins University Applied Physics Laboratory (JHU/APL) to address this specific gap [NRK⁺16, GLR⁺20]. RBC is not merely a recreational chess variant but a purpose-built experimental paradigm intended to serve as a "test bed for understanding and experimenting with autonomous decision making under uncertainty," with a particular focus on modeling the management of heterogeneous Intelligence, Surveillance, and Reconnaissance (ISR) sensors [NRK⁺16]. Whereas classical chess, a game of complete information, fails to capture the essential elements of "fog and friction" that define military and other strategic conflicts, RBC elevates these factors to be the dominant elements of the contest.

Formal Definition RBC is a two-player, zero-sum sequential game that retains the foundational elements of classical chess while introducing specific modifications to accommodate and explore imperfect information. The game is played on a standard 8×8 chessboard with traditional starting positions. The profound complexity arises from a set of crucial rule changes that systematically dismantle the assumption of perfect information.

The principal rule modification is that players cannot see their opponent's pieces. A player's knowledge of the board state is confined to the locations of their own pieces and information gathered through specific game events [GPS⁺22]. The primary mechanism for information acquisition is the *sensing action*. Before making a move, each player selects a 3×3 square region of the board to sense. The game arbiter immediately provides that player with perfect information about the contents of the chosen region. This action is entirely private; the opponent is not informed of which area was sensed, nor of the result [GPS⁺22].

Information is also conveyed through captures. When a capture occurs, both players are notified of the square on which it took place, but not the type of the capturing or captured piece. The lack of board visibility also necessitates a change in move adjudication. If a player attempts a move that is obstructed by an unseen piece, the moving piece executes the longest possible legal move along the intended path and stops. The moving player is then informed of their piece's final location. Finally, the objective is altered: the concept of "check" is removed, and a player wins by capturing the opponent's king. A draw is declared after 50 consecutive moves without a capture or pawn move. A comparative summary of these mechanics is presented in Table 2.4.

Table 2.4: A Comparative Analysis of Game Mechanics: Classical Chess vs. Reconnaissance Blind Chess.

Feature	Classical Chess	Reconnaissance Blind Chess
Board Visibility	Perfect and continuous observation of all pieces.	No direct observation of opponent's pieces. Knowledge is limited to own pieces and results of private sensing.
Sensing	Not applicable.	A mandatory, private action per turn. Player chooses a 3×3 area to observe perfectly.
Capture Information	The identity and location of both capturing and captured pieces are common knowledge.	Only the location of the capture is common knowledge. The identities of the pieces involved are not revealed.
Move Legality	Players can only make legal moves. An illegal move is not possible.	Players may attempt moves that are obstructed. The piece travels the longest possible legal path.
Game Objective	Win by checkmating the opponent's king.	Win by capturing the opponent's king. There is no concept of "check" or "checkmate."

The Information Structure of RBC The defining characteristic of RBC is its information structure. Formally, it is a two-player, zero-sum, sequential game of imperfect information. In such games, a player's policy π depends not on the true, objective state of the game, but on their *belief state*—a probability distribution over all possible true states, denoted Π_0 . The set of all possible game states consistent with a player's history h is the *information set*. In RBC, the size of this information set can become extraordinarily large, with the median of the maximum number of possible board states encountered in a competitive game numbering in the thousands [Cla21]. The primary driver of this complexity is

the private sense action. Because a player is unaware of the opponent’s sensing choices, they must reason not only about the possible locations of the opponent’s pieces but also about the opponent’s own information set. This recursive reasoning about knowledge leads to a combinatorial explosion in the number of possible game histories [GPS⁺22]. This stands in stark contrast to a game like poker, where a large, public board provides a significant anchor of common knowledge. While this vast uncertainty suggests a strategic goal of uncertainty reduction, results from NeurIPS RBC competitions indicate a more complex reality. Winning bots were not invariably those that most aggressively minimized the number of possible board states [GPS⁺22, GLR⁺20]. This suggests that the *value of information*—the quality and relevance of knowledge gained—is more critical than the raw quantity of uncertainty reduction, pointing to a deeper research challenge in evaluating information in adversarial settings.

Algorithmic and Theoretical Challenges

The unique structure of Reconnaissance Blind Chess (RBC) renders it resistant to the canonical algorithms that have achieved superhuman performance in other benchmark games, establishing it as a formidable "challenge problem" for the AI community [NRK⁺16].

Algorithms designed for perfect-information games, such as Minimax search and the Monte Carlo Tree Search (MCTS) variants that powered AlphaZero [SHS⁺18], are fundamentally unsuitable for RBC. These methods depend on the existence of a single, known game state at a specific point in time, from which to project future possibilities. However, in RBC, no such single state exists for the players; instead, they must plan over a vast belief state.

The unsolved nature of RBC has fostered a diverse ecosystem of algorithmic approaches, as showcased in the annual NeurIPS competitions. A dominant approach among these is multi-hypothesis tracking. This method involves explicitly maintaining a set of all possible board states consistent with observations. Bots such as StrangeFish2 (the 2022 winner) and Fianchetto are built on this principle. They typically couple state tracking with a powerful classical chess engine like Stockfish to evaluate moves, utilizing a value function to score each of the multitude of hypothesized boards. Decisions are then made by aggregating these evaluations, for instance, by maximizing the expected or worst-case outcome [GPS⁺22].

Another significant class of algorithms is based on Monte Carlo planning,

which adapts MCTS to the imperfect-information setting. A notable example is Châteaux, the winner of the 2020 competition. It employed Deep Synoptic Monte Carlo Planning (DSMCP), a novel algorithm that uses a compact, stochastic abstraction of information states as input to a deep neural network that guides the search [Cla21, GPS⁺22].

The sheer diversity of these methods—ranging from explicit state tracking and logical inference to deep learning and Monte Carlo search—underscores that the field is still in a rich, exploratory phase. Despite the ingenuity demonstrated, RBC remains a fundamentally open problem, solidifying its status as a vital and compelling domain for future research in artificial intelligence.

2.9 The Physical Internet

2.9.1 Conceptual Foundations and Architectural Framework

It is estimated that global freight demand will triple between 2015 and 2050 [Int19]. Substantial advancements in transport and logistics services will be necessary to accommodate this growth. The Physical Internet (PI) is a conceptual framework for an open global logistics system founded on physical, digital, and operational interconnectivity [MMB13]. As defined within the foundational literature, the Physical Internet is "an open global logistics system founded on physical, digital and operational interconnectivity through encapsulation, interfaces and protocols. It is a perpetually evolving system driven by technological, infrastructural and business innovation". It proposes an interconnected network of logistics hubs and service providers for transporting goods in standardized, modular containers, drawing inspiration from the packet transmission principles of the Digital Internet.

The Roadmap to the Physical Internet [ALI17] outlines a developmental trajectory detailing the progressive generations of Logistics Nodes from 2015 to 2040. It illustrates a strategic plan that projects the evolution of logistics through five distinct generations, each characterized by unique attributes and advancements in technology and system integration. This roadmap serves as a conceptual model that provides a strategic framework for stakeholders to optimize their operations and investments.

2.9.2 Indicators of Systemic Unsustainability

The prevailing global logistics framework, despite its crucial role in enabling international commerce and contributing to societal welfare, faces a significant and intensifying challenge to its sustainability. In a seminal contribution, Mon-treuil [Mon11] articulates the "global logistics sustainability grand challenge," contending that extant methodologies for transportation, handling, storage, and supply are fundamentally untenable from economic, environmental, and social perspectives. This challenge requires a paradigm shift, rather than incremental optimization, to mitigate a trend of increasing inefficiency and negative externalities. The economic objective is the realization of substantial gains in global productivity; the environmental objective is a significant reduction in energy consumption and greenhouse gas emissions; and the societal objective is to improve quality of life by enhancing access to goods for all populations [Mon11]. In response, the Physical Internet has emerged as a transformative concept, proposing a systemic and holistic restructuring of logistics based on the architectural principles of the Digital Internet. This concept aims to transcend the fragmented and inefficient state of current logistics, enabling a transition towards a hyperconnected and sustainable logistics paradigm [PBHM17], [TMLZ20].

The unsustainability of the current system is manifested through several documented indicators of systemic inefficiency, which have been compounded by the growth of e-commerce and rapid urbanization [Mon11]. A principal indicator is the suboptimal utilization of transport assets, where vehicles exhibit low volumetric capacity utilization and a high incidence of empty movements. This inefficiency extends to inventory management, where substantial capital is allocated to underutilized inventory, coexisting with inadequate service levels. Furthermore, infrastructure utilization is poor, with facilities being underused during off-peak periods and overstrained at others. The system is also characterized by operational deficiencies that result in unreliable intermodal transport, particularly within congested urban areas. It is marked by a notable vulnerability to disruptions and an environment where innovation is constrained by a lack of standardization and modularity, favoring marginal improvements over fundamental advancements [Mon11].

2.9.3 Core Components and Definitions

The PI architecture is defined by thirteen interrelated principles that collectively delineate the paradigm shift from traditional logistics [Mon11]. The fundamental architectural component is the principle of encapsulation, realized through world-standard, smart, and modular π -**containers** [Mon11]. Analogous to a data packet in the digital domain, the π -container abstracts the complexity of physical goods. By encapsulating products within a standardized form factor, the PI shifts the logistical focus from handling a near-infinite variety of items to managing a finite set of standard containers [Mon11]. These containers are engineered to be modular for efficient composition and decomposition, equipped with smart-tagging technology for real-time tracking, and constructed from environmentally sound materials to minimize secondary packaging requirements [Mon11].

These π -containers are processed at π -**nodes**, which constitute the logistics centers of the Physical Internet. A π -node is defined as an "open-access logistics facility that is part of the Physical Internet network and is capable of efficiently handling, storing, and transshipping π -containers" [MMB13]. These nodes are interconnected, forming a network that facilitates the unimpeded flow of goods. The system is designed to achieve universal interconnectivity, transitioning from conventional material handling to more efficient π -**container handling systems**, and from point-to-point transport to a distributed multi-segment inter-modal transport model. This infrastructure supports an Open Global Supply Web, wherein products are designed for optimal volumetric efficiency within containers, and physical movements are minimized through the digital transmission of knowledge for local materialization where feasible [Mon11].

2.9.4 Analogy with the Digital Internet

The Physical Internet is conceptualized as the logistics analogue of the Digital Internet, adapting its fundamental principles to improve logistical efficiency [DF21]. The PI is a complex network comprising diverse entities, including transportation modes and distribution centers, which must address both network reachability and operational optimization. To structure this intricate system, Montreuil [MMB13] proposed a seven-layer Open Logistics Interconnection (OLI) model, which is analogous to the Open Systems Interconnection (OSI) model in digital networking. The OLI model provides a hierarchical framework for logistics operations, spanning from the physical handling of con-

tainers to the application layer that manages orders[MBF12]. Table 2.5 details the OLI model.

Table 2.5: OLI model

No.	Layer	Functionality provided to the upper layer
7	Application	Purchase order with delivery requirements
6	Presentation	Order monitoring
5	Shipping	Shipment and π -container assignment and monitoring
4	Routing	Route assignment and monitoring for each shipment and π -container
3	Network	Network state monitoring, Route segment, assignment and monitoring
2	Link	Validation and monitoring of each flow link state and route segment move orders
1	Physical	π -container and π -means, state monitoring and move order tracking

The operational logic of the PI is dependent upon adaptive and dynamic routing, which requires novel decentralized optimization algorithms[KLF20] [NFS25]. Research has investigated agent-based models wherein PI hubs function as autonomous, decision-making agents [VTN23] [QSZ⁺20]. In such a model, an origin hub communicates with other network hubs to identify feasible subsequent transshipment points that satisfy delivery constraints[KLF20] [SNS⁺19]. The objective is to minimize a total cost function incorporating time, transport cost, and emissions [SBHK19]. To manage the computational complexity inherent in a large-scale network, such algorithms may employ operations research techniques, including breadth-first search and branch-and-bound, to identify optimal paths with efficiency. This decentralized, agent-based methodology is inherently more scalable and resilient than centralized control models, which is consistent with the distributed principles of the PI framework [FSLvM23].

2.10 Related Work in Physical Internet Research

The theoretical foundations of the Physical Internet (PI) have spurred a diverse body of research aimed at operationalizing its core concepts. This section reviews significant methodological contributions by organizing the literature into key themes: foundational architectural frameworks, operational optimization, advanced modeling techniques, and the socio-technical dimensions of PI implementation. Collectively, these works advance the understanding of how a hyperconnected logistics system can be designed, managed, and optimized [KLF21][GPD21].

2.10.1 Architectural and Operational Frameworks

A significant corpus of research has been dedicated to establishing the PI's architectural and operational frameworks. Drawing strong analogies from digital network protocols, scholars have progressively refined the PI's structure from abstract layers to concrete technical blueprints. Early work translated the Open Systems Interconnection (OSI) model into an Open Logistics Interconnection (OLI) model, defining distinct layers for physical transport, routing, and encapsulation [Mon11][MBF12]. This line of inquiry has advanced significantly, with subsequent research proposing detailed frameworks for the road-based PI based on TCP/IP functionalities [KLF21] and, more recently, developing a PI-Link Protocol to solve the fundamental challenge of network reachability [NFS25]. This progression reflects a move towards tangible, implementable protocols with a growing emphasis on decentralized architectures that prioritize data ownership and transparency through data space principles [SMM⁺24][Hof23].

Building on these architectural concepts, another research stream focuses on the operational and planning challenges within PI networks. A key area has been the optimization of logistics activities, including routing, consolidation, and resource allocation. Studies have explored dynamic and decentralized routing mechanisms to enhance network flexibility and resilience against uncertainty and disruptions [SBHK19][SM24]. The principle of modularity has also been operationalized through research on dynamic containerized consolidation, which aims to minimize handling at intermediate hubs and expedite parcel flows [KM21].

2.10.2 Modeling and Analysis Methodologies

To assess the viability and performance of PI concepts, researchers have widely employed simulation and digital twin methodologies. Simulation, in particular, has been critical for evaluating PI performance in specific contexts. For example, Ban et al. [BDG⁺21] utilized a hybrid discrete-event and agent-based simulation to model the last-mile delivery network in Cambodia, assessing the PI's potential to overcome infrastructural deficits in developing nations. Similarly, Kim et al. [KMKK21] employed a detailed simulation for a holistic assessment of an integrated urban fulfillment and delivery system. Their model encompasses inventory, fulfillment, and transportation decisions across a multi-echelon network, enabling a comprehensive analysis of system-wide performance trade-offs.

The Digital Twin, a virtual representation of a physical system, has emerged as an even more powerful tool for PI management. Cipres et al. [CCPLR21] introduced GPICS, a standardized framework that uses multi-agent simulation to construct a digital twin of a supply network, uniquely designed for real-time integration with operational PI services via APIs. While their work focuses on the framework itself, Campos et al. [CDM⁺21] addressed the prerequisite stage by outlining a rigorous, tool-agnostic methodology for developing the domain and conceptual models necessary to build a digital twin for a PI-enabled hub relay network.

2.10.3 Governance, Sustainability, and Implementation

Beyond the technical and operational facets, a growing body of work addresses the critical "soft" dimensions of PI implementation, including governance, stakeholder collaboration, and business models. Researchers underscore that the success of the PI hinges not just on technological sophistication but also on establishing trust, transparency, and fair governance mechanisms among network participants [FTP23][SMM⁺24]. This has led to the development of readiness indicators to help organizations assess their capacity for PI adoption [GSBS25]. Furthermore, the broader impacts of the PI on sustainability and the circular economy are gaining increasing attention. Studies have quantitatively assessed the potential for PI-enabled networks to reduce carbon emissions [DHS20] and have conceptualized hyperconnected circular supply chains that leverage PI principles to close material loops and enhance resource efficiency [WOL⁺23].

2.11 Capacity Management in the Physical Internet

A central operational challenge in realizing the Physical Internet (PI) is the effective management of logistics capacity. Traditional capacity management in the logistics sector often relies on static, long-term contractual agreements, which are ill-suited for the dynamic and interconnected environment envisioned by the PI [Mon11]. To operate within a hyperconnected system, service providers, particularly "Third-Party Logistics (3PL) providers"—companies that manage outsourced elements of distribution and fulfillment for other firms—must adopt more agile, data-driven methodologies. As emphasized by Boerger and Montreuil [BM20], this transition is essential for governing key resources such as warehouse space and transportation fleets. This section surveys the primary methodologies advanced in the literature for PI capacity management, with a focus on foundational conceptual frameworks and the application of sophisticated computational techniques like optimization and reinforcement learning.

2.11.1 Foundational Frameworks for Capacity Management

A seminal contribution to structuring capacity management in this new paradigm is the three-tiered analytical framework proposed by Boerger and Montreuil [BM20]. This model provides a roadmap for 3PLs to evolve from a reactive to a proactive operational posture by organizing decision-making into three distinct layers. The first, the 'Descriptive Layer', focuses on the comprehensive monitoring of historical and current operational states, such as warehouse occupancy and vehicle utilization, providing a clear view of "what has happened" and "what is happening." The second, the 'Predictive Layer', utilizes historical data and external factors to forecast future capacity demand and resource availability, addressing the question of "what will happen." The third and most advanced layer, the 'Prescriptive Layer', employs the outputs of the lower layers to recommend optimal actions and strategic decisions, such as dynamic pricing or resource allocation, to answer "what should be done."

This framework is particularly relevant for managing multi-tenant warehouse environments, which are characteristic of PI π -nodes and are subject to high demand volatility from diverse clients. The model provides a structured pathway for transforming raw data into actionable intelligence. This perspective is

enriched by the broader PI concept of the "Logistics Web" [Mon11], which conceives of capacity not as a fixed asset owned by a single firm, but as a fluid and fungible resource that can be accessed and shared across a distributed network of interconnected providers.

2.11.2 Optimization Methods for Prescriptive Analytics

The prescriptive layer of the capacity management framework has been a fertile ground for the application of mathematical optimization. These techniques provide a formal basis for solving the complex resource allocation problems inherent to the PI.

At the strategic and tactical echelons, research has focused on network design and long-term capacity planning. Many studies employ "Mixed-Integer Linear Programming (MILP)", a powerful optimization technique for problems involving both continuous variables (e.g., quantity of goods) and discrete decisions (e.g., whether to open a facility). These MILP models are used to determine the optimal location, number, and scale of PI hubs, balancing trade-offs between transportation costs, inventory holding costs, and customer service levels to design an efficient and resilient network.

At the operational level, optimization is used for real-time scheduling and dynamic resource allocation within PI facilities. A key challenge is managing throughput at "cross-docks", which function as critical π -nodes designed for the rapid consolidation and transfer of π -containers with minimal storage time. Fazili et al. [CBRT22], for example, developed an integrated model for truck scheduling and storage space allocation at cross-docks to minimize total service time and maximize facility throughput. Given the computational complexity of such real-time problems, researchers often develop "heuristics and metaheuristics"—specialized algorithms designed to find high-quality, near-optimal solutions within a very short timeframe. Furthermore, auction-based mechanisms have been proposed as a market-driven approach for dynamic capacity allocation, allowing service providers to bid for available transport or storage capacity, which aligns with the PI's vision of an open and competitive logistics market.

2.12 Conclusion

This chapter has established the foundational knowledge required for the contributions of this thesis by surveying two primary fields: the Physical Internet and Reinforcement Learning. The review began by outlining the conceptual framework of the Physical Internet, its architectural principles, and the systemic inefficiencies it aims to resolve. The specific operational challenge of Capacity Management in the Physical Internet was highlighted, as this context forms the basis for the Gym-DC framework that will be formally introduced and detailed in Chapter 5. The second half of the chapter provided an overview of Reinforcement Learning, starting from its fundamental mathematical notations and principles, and extending to advanced topics such as Reward Machines and Automated RL. The core concepts and notations of Reinforcement Learning presented here create the formal language that will be utilized in Chapter 3 to propose and analyze RL-based strategies for dynamic capacity management. Finally, the discussion on the challenges of applying RL to real-world problems motivated the need for more evaluation methods. Two complex problem domains introduced—the partially observable game of Reconnaissance Blind Chess and the hyperparameter optimization challenge in Footstep Biometrics and Genomic Modeling—serve as testbeds for the methods developed in this thesis. The empirical results from applying our approaches to these two challenging examples will be presented in Chapter 4.

Chapter 3

Uncertainty quantification for RL algorithms in POMDP environments

This chapter presents a comparative analysis of two frameworks for classifying the complexity of RL problems. The first methodology assesses complexity through a suite of established benchmark environments [DALM⁺21]. The second methodology, which is introduced in this chapter, classifies problems based on the existence of optimal policies to guide algorithm selection, particularly for the GRM approach.

The exploration extends to a rigorous analysis of the existence of optimal policies, highlighting the significance of various assumptions and methodologies in different studies. Additionally, two theoretical frameworks are presented that demonstrate conditions under which an optimal policy does not exist. An analysis of how these theorems inform the feasibility of RL approaches is then discussed.

3.1 Introduction

In reinforcement learning research, tackling emerging challenges requires careful consideration of algorithm selection, development, and evaluation. Numerous algorithms have been developed and tested against diverse benchmarks to assess their effectiveness in addressing complex real-world issues. Dulac-Arnold et al. [DALM⁺21] identify several key challenges in reinforcement learning

which are mentioned in section 2.4. As part of the effort to address these challenges, Dulac-Arnold et al. [DALM⁺21] propose a set of configurable benchmark problems, and a classification of different configurations as Easy, Medium or Hard. The settings for these three classes were presented in table 2.3. This classification provides a structured approach to evaluating the capability of algorithms in scenarios ranging from straightforward to highly complex, enabling researchers to better understand the strengths and limitations of different methods within the field. Given these considerations, when reinforcement learning is applied to a new domain, researchers can refer to existing studies to quickly identify the most suitable algorithms. They can evaluate various proposed algorithms and, if addressing a specific real-world challenge, focus on issues like large action or observation spaces. A structured Approach to Developing Reinforcement Learning Applications is illustrated in Table 3.1.

Table 3.1: Structured Approach to Developing Reinforcement Learning Applications

Step	Description
Challenge Identification	<i>Identify the challenges that need to be overcome to tackle the problem successfully</i>
Benchmark Proposal	<i>Identify specific instances for each difficulty that will allow the effectiveness of different approaches to be measured</i>
Method Development	<i>Develop tailored methods for the problem based on the challenges and difficulty level</i>

An "RL pipeline" is a structured sequence of stages designed to automate the various components involved in solving reinforcement learning tasks [PHRS⁺22] [MLM⁺23]. When evaluating a newly proposed reinforcement learning pipeline, researchers can use benchmarks with high complexity. If a particular challenge is not addressed in prior studies or if a new metric is required to measure complexity, they should consider the specific domain's distinct situations, especially at the "hard" level of complexity.

One issue that impacts the effectiveness of reinforcement learning algorithms, but not included in the above classification scheme, is the existence of a well-defined optimal policy. The theoretical justification for many algorithms relies on an assumption that an optimal policy exists and is reachable. We believe that many of the problem instances that fall into the Hard class are hard because the optimal policy either does not exist, or is impossible to reach in the search

space. If there is no optimal policy, then the selected algorithm may need to be adapted to account for this, to allow it to converge to an acceptable policy instead. This chapter focuses on the existence or otherwise of optimal policies, framing this theoretical analysis as a guide to whether an RL problem can be addressed using standard approaches or specialized ones like GRM. In section 3.2 we discuss some existing results relevant to the existence of optimal policies, and we then propose two new results stating conditions under which optimal policies do not exist. Finally, we contrast this solvability perspective with the earlier benchmark proposal by Dulac-Arnold et al. [DALM⁺21].

3.2 Existence of Optimal policy

In this section, we present four theorems from the literature that are used to establish that under certain conditions specific RL algorithms will eventually converge to an optimal policy. As part of that, these results first establish that, for the given assumptions, an optimal policy does in fact exist. We then present two new theorems, which show that for some other scenarios, there is no well-defined optimal policy. All the notations and formulas used in this chapter were presented in Section 2.2.

Optimal value function: A policy π^* is total reward optimal whenever the value function has the highest value for each state in the MDP state set or POMDP belief set[Put14]:

$$v^{\pi^*}(s) \geq v^{\pi}(s) \quad \text{for each } s \in S \text{ and all } \pi \in \Pi$$

The value function for MDP:

$$V^{\pi}(i) = \sum_{j \in S} T(i, \pi(i), j) (r(i, \pi(i), j) + \gamma V^{\pi}(j))$$

And the value function for POMDP is:

$$V^{\pi}(i) = \sum_{j \in S} T(i, \delta(i), j) [r(i, \delta(i), j)] + \sum_{o \in O} M(o, j, a) V^{\pi}(B_{a,o})$$

3.2.1 Convergence of the value function

Given a MDP (S, A, T, R) and a stationary policy δ , the sequence V_t generated by the Temporal-Difference (TD(0)) update rule:

$$V_{t+1}(i) = V_t(i) + \alpha_t(i) z_t(i) d_t$$

where $d_t = R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t)$ is the temporal-difference (TD) error, $z_t(i)$ is an eligibility trace (equal to 1 if $i = S_t$ and 0 otherwise), and $\alpha_t(i)$ is the step-size for state i .

Under policy δ , the sequence converges with probability 1 to the value function V^δ along an infinite trajectory under policy δ , as long as every state is visited an infinite number of times, and the step-sizes α_t satisfy:

$$\sum_{t=0}^{\infty} \alpha_t = \infty \quad \text{and} \quad \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

The proof is provided in [MRN05], based on the assumption that all states are visited infinitely often, and the Markov chain is ergodic.

3.2.2 Convergence of the Q function

Given a MDP (S, A, T, R) and a stationary policy δ , the sequence Q_t , generated by

$$Q_{t+1}(i, a) = Q_t(i, a) + \alpha_t(i, a) (R_t + \gamma \max_{b \in A} Q_t(j, b) - Q_t(i, a))$$

The authors in [MRN05] demonstrated that under policy δ , the sequence Q_k converges to the optimal Q^* along an infinite trajectory under policy δ , provided that every state-action pair is explored an infinite number of times and the step-sizes α_t satisfy

$$\sum_{t=0}^{\infty} \alpha_t = \infty \quad \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

$$Q^*(i, a) = \sum_{j \in S} T(i, a, j) (r(i, a, j) + \gamma V^*(j))$$

3.2.3 Convergence of SQ Learning

The SQ-Learning (Sampled Q-Learning) algorithm is a method designed for Partially Observable Markov Decision Processes (POMDPs). Starting with an initial belief state $\Pi_0 = \pi_0$, the algorithm selects N samples from the belief state space $\mathcal{S}^p$ and initializes the Q-values $Q_0(\tilde{\pi}_i, a)$ for each sample $\tilde{\pi}_i$. At each time step, it calculates an estimated Q-value $\hat{Q}^-$ by averaging the Q-values across samples, using a weighting function $\mathcal{K}(\|\pi_t - \tilde{\pi}_i\|)$. Here, $\mathcal{K}(\|\pi_t - \tilde{\pi}_i\|)$ represents a weighting kernel that depends on the distance $\|\pi_t - \tilde{\pi}_i\|$ between the current belief state π_t and each sample $\tilde{\pi}_i$; it assigns higher weights to samples closer to π_t . A kernel is a function that defines a measure of similarity or influence, often used in algorithms to give more importance to points closer to a specified target within a given space.

After selecting an action a_t according to a specified policy δ , the agent observes the reward R_t and updates the belief state to $\Pi_{t+1} = \pi_{t+1}$. It then computes the maximum future Q-value $\hat{Q}^+$ over all actions. The Q-value update uses a temporal difference term $d_t = R_t + \gamma\hat{Q}^+ - \hat{Q}^-$, where γ is the discount factor, and adjusts each sample's Q-value based on whether the action matches the selected a_t .

Assumptions:

Consider a POMDP (S, A, O, T, M, R) where the underlying Markov chain is ergodic for any stationary policy. Suppose the policy δ , used to generate the learning trajectory, assigns a non-zero probability to all actions. Furthermore, let the step-sizes α_t satisfy the conditions

$$\sum_{t=0}^{\infty} \alpha_t = \infty \quad \text{and} \quad \sum_{t=0}^{\infty} \alpha_t^2 < \infty.$$

Theorem Convergence of SQ-Learning Under the above conditions, the SQ-Learning algorithm converges to an optimal approximation of Q^* over the sample set $\{\tilde{\pi}_1, \dots, \tilde{\pi}_N\}$, as shown in [MRN05].

3.2.4 Convergence of Q-value Decision Processes

A history-based decision process P is a stochastic mapping from a history-action pair to observation-reward pairs. Formally, $P : H^* \times A \rightarrow O \times R$ where denotes a stochastic mapping. We use Q to denote an action-value function of a HDP, and

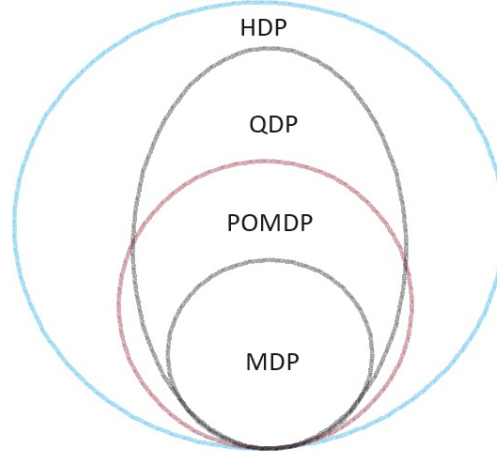


Figure 3.1: QDP in the perspective of other decision problem classes[MH18].

Q^* denotes the optimal Q-value function.

$$Q^*(h, a) := \sum_{\substack{o' \\ r'}} P(o', r' | h, a) \left(r' + \gamma \max_{\check{a}} Q^*(h a o' r', \check{a}) \right)$$

State-uniformity condition: For any action a , if any two histories h and $\hat{h}$ map to the same state s , then the optimal Q-values of the underlying HDP of these histories are the same, i.e., state-uniform; $Q^*(h, a) = Q^*(\hat{h}, a)$. It is easy to see that in this case $q^*(s, a) = Q^*(h, a)$. The state-uniformity condition is weaker than the MDP condition.

Q-Value Uniform Decision Process (QDP): A model is a QDP if it satisfies the state-uniformity condition.

Convergence in QDP Assuming the state process is ergodic and QDP, the agent learns the optimal action-value function of a QDP state-process[MH18]. These classes are illustrated in Figure 3.1.

The theorems presented in the preceding sections are crucial, as they provide the proofs for convergence and establish the conditions for the existence of an optimal policy. The remainder of this section will introduce theorems that guarantee the non-existence of an optimal policy. Finally, all the presented theorems will be compared in a concluding section.

3.2.5 Random reward functions

The preceding sections have established fundamental theoretical results demonstrating conditions under which specific reinforcement learning (RL) algorithms converge to an optimal policy. In this section, we shift focus to our contribution, wherein we examine scenarios under which an optimal policy does not exist. Given the following assumptions, an optimal policy does not exist.

1-The MDP state or POMDP belief state is fully connected

2- The reward function changes over time so that the sequence of expected values is non-monotonic and does not converge. Since the sequence is infinite and does not converge, for any time point n there will exist two other time points m and o after n such that

$$E(R(S)_n) > E(R(S)_m)$$

and

$$E(R(S)_n) < E(R(S)_o)$$

The rewards are generated by a function, which changes from one time period to the next.

After n timesteps, the interaction between the agent and the environment results in a sequence of states, actions, and rewards, representing the trajectory of the agent:

$$S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_n, A_n, R_n, S_{n+1}$$

Here, the reward is a random variable. Let $R(S)_t \equiv r_t(S_t, S_{t+1})$ denote the random reward received when transitioning from S_t to state S_{t+1} at time t . Under this assumption, the reward function has a different distribution at each time step, while the state transition dynamics in the environment remain the same.

and

$$E(R(S)_n^{\pi_t}) > E(R(S)_o^{\pi_t})$$

In this notation, $E(R(S)_t^{\pi_t})$ represents the expected reward at state S under a single, time-dependent policy π_t , reflecting that the reward expectations shift as t varies. This notation emphasizes that, even under a consistent policy framework, the expected reward changes over time due to the non-stationary nature of the environment.

Proof.

The main idea regarding the proof is as follows: Based on the assumptions the expected value of the reward value, represented as a random variable, changes across different timesteps. Referring back to the definition of an optimal policy, the value function for each state should be maximized for it to align with the optimal value function. Based on the assumption about the fully connected state set, a contradiction arises between the existence of an optimal policy and two assumptions about the RL environment described above.

V^δ is the value function for policy δ which maps states to real values:

$$V^\delta : S \rightarrow \mathbb{R}$$

The value of state i under policy δ is given by:

$$V^\delta(i) = \mathbb{E}_\delta \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \mid S_t = i \right]$$

where R_{t+k} is the reward received at time $t+k$, given by

$$R_{t+k} = r(S_{t+k}, \delta(S_{t+k}), S_{t+k+1}),$$

and $\mathbb{E}_\delta$ denotes the expected value operator associated with the transition probabilities defined by policy δ .

Expanding $V^\delta(i)$, we get:

$$V^\delta(i) = E[R(i, \delta(i))] + T(i, \delta(i), j) \gamma V^\delta(j)$$

Let us assume that δ_t represents the optimal policy at time t . Let $E(R(S)_n^{\delta_t})$ denote the expected value of the reward generated under this policy at time

t , and $E(R(S)_n^{\delta_m})$ denote the expected value of the reward generated by this policy at time m . Based on the definition of an optimal policy, the value function of this policy δ should have the highest value at all time steps in comparison to all other policies that exist and the following equation should be satisfied :

$$V^\delta(s) \geq V^\pi(i), s \in S$$

$$E[R(i, \delta(i))] + T(i, \delta(i), j) \gamma V^\delta(j) \geq$$

$$E[R(i, \pi(i))] + T(i, \pi(i), j) \gamma V^\pi(j)$$

Based on assumption 3, there exists a time m such that the expected value of the reward at time m is less than that at time t . Furthermore, according to assumption 2, there exists a policy whose expected value of rewards, generated by its sequence, is higher than that of policy δ at time step m . Considering both assumptions, there exists a policy π that satisfies this inequality for state i :

$$E[R(i, \delta_m(i))] < E[R(i, \pi(i))] \text{ and } T(i, \delta_m(i), j) = T(i, \pi(i), j)$$

Then:

$$T(i, \delta_m(i), j) \gamma V^{\delta_m}(j) < T(i, \pi(i), j) \gamma V^\pi(j)$$

If the value function for a given state is lower than that of another policy at any time step, it implies that a consistent optimal policy does not exist. Then $V^{\delta_m}(j) < V^\pi(j)$ which is a contradiction with the definition of optimal policy for state j .

3.2.6 Random reward functions and ε optimal policies

Given the assumptions in section 3.2.5, ε optimal policy does not exist.

While Section 3.2.5 established that a **strictly** optimal policy cannot exist under our assumptions, one might naturally ask if an approximate solution is still possible. In this section, we extend our previous result to prove that even an ε optimal policy does not exist under these same non-stationary conditions.

Proof.

The main idea regarding the proof is as follows: Based on the assumptions the expected value of the reward value, and optimal policy and using Markov's inequality , a contradiction arises between the existence of an ε optimal policy and the foundational assumption regarding the class features.

Markov's inequality:

If X is a non-negative random variable, then for all $a > 0$, which is called Markov inequality:

$$P(X \geq a) \leq \frac{E[X]}{a}$$

$$1 - P(X \geq a) \geq 1 - \frac{E[X]}{a}$$

$$P(X < a) \geq 1 - \frac{E[X]}{a}$$

Based on the definition of an ε -optimal policy, if we consider δ as the optimal policy, then there exists a policy that satisfies this condition at all time steps.

$$V^\delta(j) - V^\pi(j) < \varepsilon$$

Let us define the random variable.

$$\begin{aligned} X &= |V^\delta(j) - V^\pi(j)| \\ &= \left| E[R(i, \delta(i))] + T(i, \delta(i), j) \gamma V^\delta(j) \right. \\ &\quad \left. - E[R(i, \pi(i))] - T(i, \pi(i), j) \gamma V^\pi(j) \right| \end{aligned}$$

As in theory 5, by using assumptions 2 and 3, there exists a time m and a policy π that satisfy this inequality for state i :

$$E[R(i, \delta_m(i))] > E[R(i, \pi(i))] + a$$

and

$$T(i, \delta_m(i), j) = T(i, \pi(i), j)$$

then :

$$M = E[R(i, \delta_m(i))] + T(i, \delta_m(i), j) \gamma V^{\delta_m}(j)$$

$$N = E[R(i, \pi(i))] - T(i, \pi(i), j) \gamma V^{\pi}(j)$$

$$P(|M - N| < a)$$

$$\leq 1 - \frac{E[E[R(i, \delta_m(i))] - E[R(i, \pi(i))]]}{a}$$

Then:

$$p(|V^{\delta_m}(j) - V^{\pi}(j)|) < 0$$

which is a contradiction with the assumption and probability axioms.

Then: $p(|V^{\delta}(j) - V^{\pi}(j)|) < 0$ which is a contradiction with the assumption and probability axioms.

3.3 Guide for Problem Solvability

In discussing uncertainty within machine learning models, aleatoric and epistemic uncertainties are two primary types. Aleatoric uncertainty (also known as statistical or irreducible uncertainty) arises from inherent randomness or variability in the system, making it resistant to reduction even with more data. In contrast, epistemic uncertainty is due to a lack of knowledge about the model or system parameters, making it a reducible uncertainty that can be minimized through additional data or improved modeling efforts [HW21].

Considering the definitions of aleatoric and epistemic uncertainty, along with the assumptions in the theorems, the reward value, as a random variable, is

dominated by aleatoric uncertainty due to its variation over different periods. This prevents the agent from establishing any correlation between its belief state and the reward random variable over an infinite timeline. However, in Theorems which were presented in sections 3.2.1, 3.2.2, 3.2.3 and 3.2.4, the reward value as a random function is subject only to epistemic uncertainty, stemming from the agent’s limited knowledge of the environment.

Based on this understanding, problems can be categorized into three distinct classes:

- **Solvable (Optimal Policy Exists):** Problems whose features allow the proof of the existence of an optimal policy. Note that while solvable in theory, finding this policy may still be practically difficult.
- **Unknown Solvability:** Problems for which the existence of an optimal policy is not yet proven, however, it is not possible to prove that the optimal policy does not exist.
- **Strictly Unsolvable (No Optimal Policy):** Problems whose features allow the proof that an optimal policy does not exist.

3.4 Implications for Algorithm Selection and GRM

In the Real-world Benchmarks approach, reinforcement learning algorithms are tested in settings categorized as easy, medium, or hard by adjusting environment parameters (as seen in Table 2.3). The approach taken in this thesis provides an alternative perspective: assessing problem feasibility based on theoretical optimality criteria. Rather than offering a general complexity taxonomy, this analysis serves as a guide to determine whether an RL problem can be addressed using specific methods, such as the GRM approach. In particular:

- **Solvable problems:** Based on theorems in Sections 3.2.1, 3.2.2, 3.2.3, and 3.2.4, an optimal policy exists for certain problems with epistemic uncertainty. In these scenarios, RL algorithms or the GRM approach can be confidently applied with the theoretical guarantee that an optimal solution exists, even if it is computationally difficult to find.
- **Unknown problems :** The existence of an optimal policy remains unproven. Application of the GRM approach here requires empirical validation, as theoretical convergence cannot be strictly guaranteed based on current literature.

- **Unsolvable problems** : Problems that align with the theorems in Section 3.2.5, involving specific cases of aleatoric uncertainty, lack a well-defined optimal policy. Consequently, these problems are unsolvable in the traditional RL optimization sense. Algorithms expecting convergence to a strict optimal policy will fail. In these instances, the problem must be redefined, or approaches must be adapted to target acceptable, rather than optimal, policies.

In summary, while benchmark approaches categorize practical difficulty based on environment settings, this theoretical analysis defines fundamental solvability. This provides practitioners with a necessary foundation to evaluate whether a problem can be validly addressed using the GRM approach before committing computational resources to finding a solution.

3.5 Conclusion

We introduced a complexity classification based on the existence of optimal policies, demonstrating conditions under which optimal policies do or do not exist. This classification was further supported by two new theorems proving the non-existence of optimal policies under specific conditions. Our findings emphasize the necessity for a structured approach to RL research, integrating theoretical insights with practical benchmarks to address real-world challenges effectively. By categorizing RL problems according to their complexity and providing tailored solutions, this work aims to facilitate more robust and efficient reinforcement learning applications. Future research should continue to refine these classifications and explore new benchmarks, ensuring that RL systems can be reliably applied to increasingly complex and dynamic environments. Ultimately, the goal is to advance the field of RL by developing algorithms that are both theoretically sound and practically effective in diverse application domains.

Chapter 4

Generative reward machine for RL problems

4.1 Introduction

In this chapter, we introduce the concept of the *Generative Reward Machine* (GRM). An effective strategy for enhancing RL involves exposing the internal structure of the reward function to the agent, enabling more sample-efficient learning [IKVM22]. RMs have been proposed as a formal, automata-based representation of reward functions to achieve this. However, traditional RMs rely on clear, high-level events from the environment, a condition that is often unmet in real-world scenarios characterized by partial observability and noise, especially in the type of challenges highlighted as class of hard problem in section 3.2.5. The primary objective of GRM is to adapt and extend the reward machine concept to address these problems. We first formally define and explain the foundational idea of the generative reward machine. Subsequently, we explore the implementation and evaluate the performance of GRM across complex problem domains of Reconnaissance Blind Chess, OGB and a footstep recognition benchmark.

4.2 Generative Reward Machines

The core concept of the GRM is to create a new mapping or an intermediate, abstract representation of the environment’s state. This can be approached in two ways, chosen based on the problem. The first method uses a direct mapping to

transform environment states into a specific, predefined state space. The second involves an estimator function that maps noisy, partial observations from the environment to specific states in the estimated state space. A new mapping of state set, or an *estimator* function, maps noisy, partial observations from the environment to a specific state. To address the problem of non-stationary rewards in hard problems which are mentioned in section 3.2.5, we use an estimator or mapping function to translate each original state into a corresponding state. In a fully observable MDP, a designer with complete knowledge of the system can create an effective reward machine. However, for problems which are in the hard class as mentioned in section 3.2.5, this is impossible due to complex reward functions and aleatoric uncertainty. Our solution is to use limited information about these hard problems to design a two-part system: first, a mapping function or estimator for the state, and second, a new, specialized reward machine. Together, these components try to keep the rewards generated for the states stationary. Essentially, limited knowledge which is available is used to engineer a system that bypasses the conditions found in the hard problems. The GRM aims to approximate a RM that renders the environment Markovian over the combined observation and machine state—in scenarios where inherent noise make a true RM unattainable.

The online operational procedure of an agent employing a GRM is as follows :

1. The agent resides in an unknown true world state (ts).
2. The agent makes a partial observation of the world, resulting in an observed state (os).
3. The estimator function processes or a mapping function the observed state (os) to produce an estimated state (es).
4. The Generative Reward Machine is in a specific internal state, u , which has an associated reward function, r_u .
5. The agent selects and applies an action, a .
6. The environment transitions the agent to a new true state, ts' .
7. The agent observes the new observed state, os' .
8. The estimator function or mapping function computes the new estimated state, es' , based on the previous estimated state (es), the action (a), and the new observation (os').

9. The GRM applies its reward function, r_u , to the transition that occurred in the *estimated space*—from es to es' —to generate a reward signal.
10. A labelling function analyzes the transition in the estimated space (from es to es') and reports a set of abstract propositional symbols, b , describing the outcome.
11. The GRM's internal finite state machine uses the symbols b as input to transition from its current state u to a new state u' . The process then repeats from the new comprehensive state (ts', os', es', u') .

4.2.1 Comparison: Reward Machine vs. Generative Reward Machine

While a standard reward is simply the scalar feedback signal received by an RL agent, a Reward Machine (RM) provides a formal, automata-based structure for these reward functions, typically relying on clear events from the true environment state. The Generative Reward Machine differs fundamentally by operating on an abstracted, estimated state representation rather than the true state. The GRM is termed 'generative' because it actively generates this intermediate state space and its corresponding stationary reward signals using an estimator function, allowing the agent to function effectively despite partial observability and inherent noise.

The fundamental distinction between a standard Reward Machine and a Generative Reward Machine lies in the state representation upon which they operate. The GRM introduces a layer of abstraction—the estimated state—to manage environmental uncertainty. Table 4.1 table provides a component-wise comparison of the two frameworks.

Table 4.1: Comparison of RM and GRM Components

Component	RM [IKVM22]	GRM	Rationale for Difference
State Basis	Operates directly on environment states (S) or observations (O) .	Operates on an abstracted, <i>state</i> (E) derived from observations .	To create a stable, discrete state representation for the FSM to operate on, even when observations are noisy or partial.
State-Reward Function (δ_r)	Maps reward functions over transitions in the true environment state space ($S \times A \times S$).	Maps reward functions over transitions in the <i>designed state space</i> ($E \times A \times E$) .	uses the reward signal as input of new reward function with the abstracted space where the agent's policy is being learned, providing more consistent feedback.
Labelling Function (L)	Assigns propositional symbols based on transitions in the true environment ($S \times A \times S$).	Assigns propositional symbols based on transitions in the <i>designed state space</i> ($E \times A \times E$) .	The FSM transitions must be triggered by events in the same abstract space that the agent is reasoning about.
Truth Assignment (σ_t)	The truth values of propositional symbols are determined by the current state of the true environment.	The truth values of propositional symbols are determined by the current <i>estimated state</i> .	Follows logically from the shift in the operational basis from the true environment to the estimated representation.
Primary Objective	To find a state representation that makes the problem Markovian over the joint space ($S \times U$), enabling optimal policy learning.	To <i>approximate</i> a Markovian representation when one is not feasible, making the problem tractable for RL algorithms .	Provides a practical solution for real-world applications where perfect state information is unavailable.

4.3 The GRM Framework for Reconnaissance Blind Chess

The rules for Reconnaissance Blind Chess are described in section 2.8. The foundational RBC bot architecture is based on a Multiple Hypothesis Tracker. Unlike traditional chess engines that operate on a single, known board state, an RBC agent must maintain a belief state over all possible board configurations consistent with the game's history. The Multiple Hypothesis Tracker manages a set, or distribution, of these hypothetical board states. At each turn, this set is updated through a two-stage process:

1. **Expansion:** For each hypothesis board in the current set, all possible legal moves for the opponent are generated, creating a new, larger set of potential subsequent states.
2. **Filtering:** This expanded set is pruned based on observations from the game engine. These observations include the agent's own sense results, the outcome of its attempted move, and information regarding captures. Any hypothesis inconsistent with these observations is eliminated from the belief state.

The agent's decision-making for both sensing and moving is therefore performed over this distribution of possible realities, rather than a single deterministic state. The performance of the RBC bot is tuned by a specific set of hyperparameters, the details of which are provided in Appendix 8.4, this section focuses on three that are particularly influential in shaping the bot's strategic behavior: those controlling its aggression, predictability, and exploration. Below is the list of these three items:

- **max_score_factor:** This functions as an aggressiveness or gambling factor. A higher value encourages the bot to pursue moves that have a high potential reward, even if they are risky and the best-case scenario is unlikely to occur.
- **threshold_score:** This parameter adds variety to move selection to prevent the bot from becoming predictable. It allows the agent to choose from a pool of moves, as long as their score is within a certain margin of the best-scoring option.
- **sampling_exploration_coef:** This value directly influences the UCT (Upper Confidence bounds applied to Trees) algorithm. A higher coefficient

encourages the agent to explore moves it has considered less often, preventing it from prematurely settling on a suboptimal strategy.

4.3.1 RL for RBC

We frame the problem of selecting the optimal hyperparameters as a Reinforcement Learning task. The objective is to learn a policy, $\pi(s)$, that maps a given state, s , to an optimal action, a . In the RL approach, the state, reward, and action are defined as follows:

- **State (s):** The state space S represents the initial strategic configuration of the bot, defined by the assigned player color and the chosen opening strategy. Formally, $S = C \times \Omega$, where:
 - $C = \{\text{White, Black}\}$ represents the assigned color ($|C| = 2$).
 - $\Omega = \{O_1, O_2, O_3, O_4, O_5, O_{\text{other}}\}$ represents the strategy pool, consisting of 5 specific predefined openings (e.g., Sicilian Defense, Queen's Gambit) plus a catch-all "Other" state used as a fallback if the bot deviates from the main five ($|\Omega| = 6$).

Thus, the total size of the state space is exactly $|S| = |C| \times |\Omega| = 2 \times 6 = 12$ distinct states. The bot chooses its opening state $s \in S$ at the start of the match based on its assigned color and intended strategy.

- **Action (a):** The action space is a discrete set of pre-defined strategic "personalities," where each personality corresponds to a specific vector of hyperparameter values.
- **Reward (R):** The reward signal for the agent is calculated with the following scoring: win = +1, draw = 0, and loss = -1.

Based on the description of the environment, the pseudocode for the training algorithm using Q-learning is presented in Algorithm 2.

Algorithm 2 Q-Learning for in RBC

- 1: Let $C = \{\text{White}, \text{Black}\}$ be the set of assigned colors ($|C| = 2$).
 - 2: Let $\Omega = \{O_1, O_2, O_3, O_4, O_5, O_{\text{other}}\}$ be the set of opening strategies ($|\Omega| = 6$).
 - 3: Let $S = C \times \Omega$ be the state space, yielding exactly $|S| = 12$ discrete states.
 - 4: Let A be the discrete set of strategic hyperparameter vectors (actions).
 - 5: Initialize Q-table $Q(s, a) \leftarrow 0$ for all $s \in S, a \in A$.
 - 6: Initialize learning rate $\alpha \in (0, 1]$.
 - 7: Initialize exploration rate $\epsilon > 0$.
 - 8: **for** episode = 1 to MAX_EPISODES **do**
 - 9: Receive assigned color $c \in C$ for the current match.
 - 10: Select intended opening strategy $\omega \in \Omega$.
 - 11: Form initial state $s \leftarrow (c, \omega)$.
 - 12: Choose action $a \in A$ for state s (e.g., using an ϵ -greedy strategy).
 - 13: Play a single game utilizing opening ω and hyperparameter vector a .
 - 14: Receive terminal reward $R \in \{+1, 0, -1\}$ (win, draw, loss).
 - 15: Update $Q(s, a) \leftarrow Q(s, a) + \alpha[R - Q(s, a)]$.
 - 16: **end for**
 - 17: **return** The final policy.
-

4.3.2 The GRM Framework for RBC

To overcome these limitations, we propose a more sophisticated, two-phase framework. The first phase involves constructing an estimator to classify opponent behavior, and the second phase uses a Generative Reward Machine (GRM) to structure the RL training process for learning a hyperparameter-selection policy.

Phase 1: Constructing the Opponent Skill Estimator The foundation of our adaptive approach is the ability to accurately classify an opponent’s skill level in real time. We construct an estimator for this purpose, which consists of a bank of 20 distinct neural networks.

- **Training:** Each neural network is an autoencoder trained offline, exclusively on game histories generated by an opponent of a specific skill level (from 1 to 20). A game history is represented as a time-series of the agent’s belief state size, which serves as a proxy for the agent’s uncertainty.
- **Runtime Estimation:** During a live game, the evolving history of the

agent's uncertainty is fed into all 20 trained autoencoders. The skill level corresponding to the autoencoder with the lowest reconstruction error is selected as the current estimation of the opponent's class.

This estimator provides the state input for the subsequent RL training phase.

Phase 2: GRM-based Policy Learning With the estimator established, we re-define the learning problem within a GRM framework to provide a more stable and efficient learning signal.

In the GRM framework, the state, reward, and action are defined as follows:

- **State (s):** The state is the opponent's skill level, $s \in \{1, \dots, 20\}$, as classified by the pre-trained estimator.
- **Action (a):** The action space is a discrete set of pre-defined strategic "personalities," where each personality corresponds to a specific vector of hyperparameter values.
- **Reward (R):** To counteract the stochasticity of RBC, the reward is not based on a single game. The agent is evaluated over a batch of 100 games against an opponent bot whose skill set is fixed at a specific level. The reward signal for the agent is the average outcome of this batch, calculated with the following scoring: win = +1, draw = 0, and loss = -1. This batch-based reward calculation is performed against bots of varying skill levels.

Based on the description of the environment, the pseudocode for the training algorithm using GRM is presented in Algorithm 3.

4.3.3 Analysis of Algorithm Performance in the 2022 NeurIPS Tournament

Figure 4.1 illustrates the performance evolution of competing algorithms in the Reconnaissance Blind Chess tournament, quantified using Bayes Elo ratings. A Bayes Elo rating is a statistical approach for assessing the relative skill level of participants. Unlike traditional Elo systems that provide a single point estimate, the Bayesian method calculates a probability distribution for each agent's true strength, represented in the chart by the central line (the estimate) and the surrounding shaded area (the 95% confidence interval). This interval signifies the range within which the agent's true skill level is believed to lie with 95% probability. The term "excluding forfeits" indicates that the ratings are calculated

Algorithm 3 Training with GRM for RBC problem

-
- 1: Let $\mathcal{E}$ be the pre-trained opponent skill estimator.
 - 2: Let $S = \{1, \dots, 20\}$ be the set of opponent skill levels (states).
 - 3: Let A be the discrete set of strategic vectors (actions).
 - 4: Initialize Q-table $Q(s, a)$.
 - 5: Initialize reward history $H(s, a) \leftarrow []$ for all state-action pairs.
 - 6: Initialize visit counts $N(s, a) \leftarrow 0$ for all state-action pairs.
 - 7: **for** episode = 1 to MAX_EPISODES **do**
 - 8: Select a state $s \in S$ to train on .
 - 9: Choose action $a \in A$ for state s .
 - 10: Play a batch of $N_{\text{games}} = 100$ games against an opponent of skill s
 - 11: Let R_{batch} be the average outcome of the batch.
 - 12: Append R_{batch} to reward history $H(s, a)$.
 - 13: $N(s, a) \leftarrow N(s, a) + 1$.
 - 14: $R_{\text{avg}} \leftarrow \text{mean}(H(s, a))$.
 - 15: Update $Q(s, a) \leftarrow R_{\text{avg}}$.
 - 16: **end for**
 - 17: **return** The final greedy policy $\pi^*(s) = \arg \max_a Q(s, a)$.
-

based only on games that concluded normally (win, loss, or draw), omitting results from games terminated by technical failures such as software crashes or connectivity loss. This ensures the ratings are a purer reflection of an algorithm’s strategic prowess rather than its operational stability. The chart plots these refined Elo ratings against the number of tournament rounds, showing how the skill estimates for each agent evolved as more data became available.

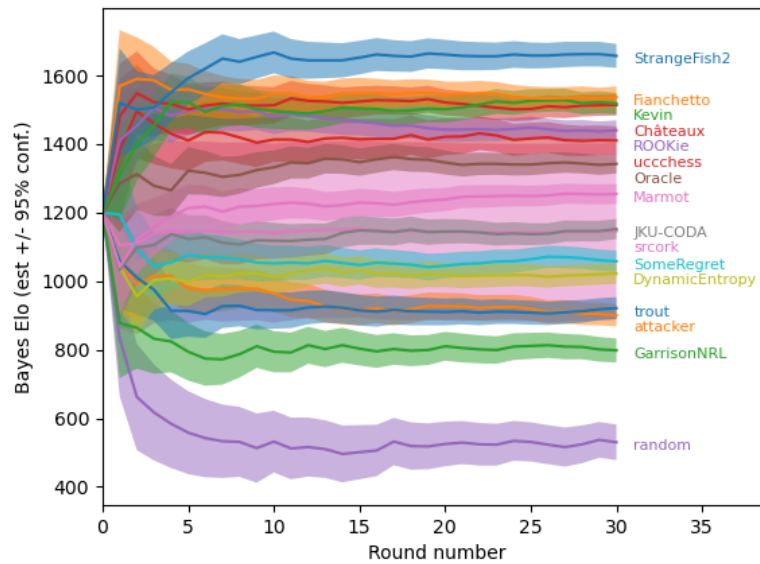
The two bots evaluated in the tournament are *srcork*, which is trained using Q-learning as mentioned in Algorithm 2, and *uccchess*, which is trained using the GRM framework as mentioned in Algorithm 3. The average Elo scores between the RL algorithm and the GRM framework are mentioned in Table 4.2.

Based on the performance trajectories shown in Figure 4.1, the GRM-trained ‘uccchess’ ranks below the top cluster of agents (*StrangeFish2*, *Fianchetto*, *Kevin*, *Châteaux*, *ROOKie*, and *Oracle* [GPS⁺22]).

Although this figure does not demonstrate a strict Elo advantage, “uccchess” proved robust; despite technical server issues requiring the removal of several tournament games, it still secured a position among the top bots and was selected in the final ranking. The methods used in these bots are described in Table 8.6.

Table 4.2: Comparison of Average Elo scores between the RL algorithm and GRM framework in the tournament, excluding forfeits.[GPS⁺22]

#	Algorithm	Average Elo
1	GRM	1365
2	RL	1190

Figure 4.1: Bayes Elo ratings and 95% confidence intervals as the tournament progressed, excluding forfeits.[GPS⁺22]

4.4 The GRM Framework for Biometric Footstep Recognition

The UNB StepUP-P150 dataset, a benchmark for Footstep Biometrics, and its Evaluation Protocols in Biometric Systems are described in Section 2.6. The problem of selecting hyperparameters for the Deep Learning architectures in this benchmark may be considered as a sequential decision-making problem and RL agent can be employed to find better hyperparameter settings for the model.

4.4.1 Problem modeling

The problem is the need to select from a variety of models and hyperparameters for training on this benchmark, such as the architectures detailed below.

$$Models = \{ViT, Swin, ConvNeXt, R(2+1)D\}$$

For each model, a list of **hyperparameters** must be assigned appropriate values before training. These hyperparameters include [PyT25] : **optimizer**, **scheduler**, **lr** (learning rate), **batch_size**, **layer_sizes**.

The selected model and hyperparameters will be used for training. During the training process, a detailed log will be generated for each epoch, capturing various model training parameters. The training log recurrently records two critical performance signals that are used in the final score calculation:

Critical Performance Signals

- **Batch Loss:** The error score a model gets after processing a single, small group of training data (a batch) and making one update to its parameters.
- **Average Loss:** The average error score the model achieved across the entire training dataset after completing one full pass through it (one epoch).

The efficacy of a selected hyperparameter configuration is quantitatively assessed via a scalar function. This score integrates two primary components derived from the complete training log: the overall mean loss (P_1) and the training stability (P_2).

The formal representation of the reward score is given by a weighted linear combination, where P_1 and P_2 are penalties derived from the training history:

$$\text{Score} = w_1 \cdot P_1 + w_2 \cdot P_2$$

Where:

- $P_1 = \text{MeanLoss}$, defined as the average of all epoch avg_loss values.

$$P_1 = \frac{1}{N_{\text{epochs}}} \sum_{i=1}^{N_{\text{epochs}}} \text{avg_loss}_i$$

- $P_2 = \text{StdDevLoss}$, defined as the standard deviation of all individual batch_loss values.

$$P_2 = \sqrt{\frac{1}{N_{\text{batches}}} \sum_{j=1}^{N_{\text{batches}}} (\text{batch_loss}_j - \text{MeanLoss}_{\text{all batches}})^2}$$

- $w_1 = -1$ and $w_2 = -1$ are the suggested penalty weights for mean loss and stability, respectively.

4.4.2 RL Framework

The reinforcement learning framework for this problem is defined as follows:

- **State Space:** The set of selectable model architectures:

$$\mathcal{S} = \{\text{ViT}, \text{Swin}, \text{ConvNeXt}, \text{R}(2+1)\text{D}\}$$

- **Action Space:** The vector of hyperparameter categories that define a configuration:

$$\Lambda = (\text{"optimizer"}, \text{"scheduler"}, \text{"lr"}, \text{"batch_size"}, \text{"layer_sizes"})$$

The reward signal for the RL agent is the negative value of the average loss. The RL-based HPO process is described in Algorithm 4.

Algorithm 4 Training with RL algorithm for Biometric Footstep Recognition

-
- 1: Initialize Q-table $Q(s, \lambda)$ for all states $s \in \mathcal{S}$ and hyperparameter actions $\lambda \in \Lambda$.
 - 2: **for** episode = 1 to MAX_EPISODES **do**
 - 3: Select an architecture state $s \in \mathcal{S}$.
 - 4: Choose action λ from $Q(s, \cdot)$.
 - 5: Train model s with hyperparameters λ for a limited number of epochs.
 - 6: Calculate the performance signal from the training log:
 $P_1 = \text{MeanLoss}$
 - 7: Compute the direct, uncalibrated reward signal:
 - 8: $R \leftarrow -P_1$
 - 9: Update the Q-table:
 - 10: $Q(s, \lambda) \leftarrow Q(s, \lambda) + \alpha (R - Q(s, \lambda))$
 - 11: **end for**
 - 12: **return** The optimal hyperparameter policy $\pi^*(s) = \arg \max_{\lambda} Q(s, \lambda)$.
-

4.4.3 The GRM Framework

As listed in previous Section, when a vector of model and hyperparameters are selected and training begins, a training log is generated for that vector. Based on this log, a score can be calculated as a reward signal. Due to a range of uncertainties, including non-deterministic GPU operations that occur during the training of deep learning algorithms on GPUs, variations in model performance based on hyperparameters, and other sources of uncertainty, this problem inherently contains a significant amount of noise.

Within the GRM framework, the unknown true environment state is conceptually defined as the one encompassing full observability of all noise sources. Since numerous sources of uncertainty remain unobserved—even when the training model and its hyperparameter are considered as a single observable component—a mapping function must be introduced. This function is essential for establishing the GRM framework; it maps the observable state to the model used for training. Thus, the new state in the GRM is represented by the Model itself, derived via this mapping function. The state space and action space in GRM framework for this problem are defined as below:

- **GRM State Space** The set of selectable model architectures:

$$\mathcal{S} = \{\text{ViT}, \text{Swin}, \text{ConvNeXt}, \text{R}(2+1)\text{D}\}$$

- **GRM Action space** The vector of hyperparameter categories that define a configuration:

$$\Lambda = (\text{"optimizer"}, \text{"scheduler"}, \text{"lr"}, \text{"batch_size"}, \text{"layer_sizes"})$$

GRM Reward function

Based on the defined action and state space the reward function for the GRM will be defined based on the model architecture. For each state (model architecture) $s \in \mathcal{S}$, 15 initial training runs are executed. For each of these 15 runs, a set of hyperparameters $\lambda \in \Lambda$ is randomly selected. For the k -th run of model s , the following data points are collected:

1. Penalty Signals $P_1^{(k)}$ and $P_2^{(k)}$: These are calculated from the avg_loss and batch_loss recorded in the training log, as defined previously:

$$P_1^{(k)} = \text{MeanLoss} \quad \text{and} \quad P_2^{(k)} = \text{StdDevLoss}$$

2. True Performance $\text{Perf}^{(k)}$: This is the actual efficacy of the model after training, evaluated on a held-out set of samples from the dataset. A higher value indicates better performance.

For each state s , the collected data (15 pairs of penalty signals and true performance values) are used to calibrate the optimal weights $\mathbf{w}_s = (w_{1,s}, w_{2,s})$. The reward estimator is designed to linearly map the penalty signals to the true performance:

$$\text{Perf} \approx w_{1,s} \cdot P_1 + w_{2,s} \cdot P_2$$

The calibration of $\mathbf{w}_s$ for model s is achieved by solving a linear regression problem that minimizes the squared error between the estimated reward and the true performance over the 15 data points. The resulting coefficients $\mathbf{w}_s = (w_{1,s}, w_{2,s})$ are the state-specific penalty weights. The GRM-based HPO process is formalized in Algorithm 5. This algorithm first learns an architecture-specific reward model and then uses that model to guide the search for optimal hyperparameters in a subsequent online phase.

Algorithm 5 Training with GRM for Biometric Footstep Recognition

Phase 1: Offline Reward Estimator Calibration

- 1: Let $\mathcal{S} = \{\text{ViT}, \text{Swin}, \text{ConvNeXt}, \text{R}(2+1)\text{D}\}$
- 2: Initialize dataset $D_s \leftarrow \emptyset$ for each state $s \in \mathcal{S}$.
- 3: Λ is the set of actions.
- 4: **for** each architecture $s \in \mathcal{S}$ **do** **do**
- 5: **for** $k = 1$ to 15 **do** **do**
- 6: Sample a random hyperparameter configuration $\lambda^{(k)} \in \Lambda$.
- 7: Train model s with hyperparameters $\lambda^{(k)}$ to completion.
- 8: Calculate Penalty Signals $P_1^{(k)} = \text{MeanLoss}$ and $P_2^{(k)} = \text{StdDevLoss}$.
- 9: Obtain True Performance $\text{Perf}^{(k)}$ from the held-out set.
- 10: Store penalty vector $\mathbf{P}^{(k)} = (P_1^{(k)}, P_2^{(k)})$.
- 11: $D_s \leftarrow D_s \cup \{(\mathbf{P}^{(k)}, \text{Perf}^{(k)})\}$.
- 12: **end for**
- 13: Perform **Linear Regression** on D_s to find the optimal weight vector $\mathbf{w}_s = (w_{1,s}, w_{2,s})$ by minimizing:

$$\sum_{k=1}^{15} (\text{Perf}^{(k)} - (w_{1,s} \cdot P_1^{(k)} + w_{2,s} \cdot P_2^{(k)}))^2$$

- 14: **end for**

Phase 2: Online Q-Learning-based Hyperparameter Search

- 15: Initialize Q-table $Q(s, \lambda)$ for all states $s \in \mathcal{S}$ and $\lambda \in \Lambda$.
 - 16: **for** episode = 1 to MAX_EPISODES **do**
 - 17: Select an architecture state $s \in \mathcal{S}$.
 - 18: Choose action λ from $Q(s, \cdot)$.
 - 19: Train model s with hyperparameters λ .
 - 20: Calculate Partial Penalty Signals P_1^{partial} and P_2^{partial} .
 - 21: $R^{\text{est}} \leftarrow w_{1,s} \cdot P_1^{\text{partial}} + w_{2,s} \cdot P_2^{\text{partial}}$.
 - 22: Update $Q(s, \lambda)$.
 - 23: **end for**
 - 24: **return** The optimal hyperparameter policy $\pi^*(s) = \arg \max_{\lambda} Q(s, \lambda)$.
-

4.4.4 Results

Table 4.3 illustrates the performance evaluation of the RL algorithm detailed in Algorithm 4 compared to the GRM framework detailed in Algorithm 5.

Table 4.4 illustrates the performance evaluation of competing algorithms in the

Table 4.3: Comparison of Weighted Total scores between the RL and GRM on the Biometric Footstep Recognition Benchmark

#	Algorithms	EER	FMR100	ACC	BACC	FNMR	FMR
1	GRM	10.77	59.63	88.51	88.94	10.00	12.13
2	RL	13.13	77.78	86.57	86.85	12.47	13.84

"StepUP Competition for Biometric Footstep Recognition"[LMP⁺25]. A detailed explanation of the TI-MFBO and ECCO methods is provided in Appendix 8.7. The results were compared based on the different metrics defined in Section 2.6. According to the metric definitions, a better model is characterized by lower FMR, FNMR, FMR100, and EER, and higher BACC and AAC. In accordance with the competition rules, the final ranking was established by sorting based on the EER, where an algorithm with a lower EER signifies superior performance in the final evaluation. The GRM framework achieved the highest performance in the International StepUP Competition for Biometric Footstep Recognition by the EER value of 10.77 .

Table 4.4: Comparison of performance metrics for different teams in International StepUP Competition for Biometric Footstep Recognition.

#	Team Name	EER	FMR100	ACC	BACC	FNMR	FMR
1	GRM	10.77	59.63	88.51	88.94	10.00	12.13
2	TI-MFBO	11.33	67.34	87.61	88.37	9.73	13.53
3	ECCO	11.50	67.12	86.99	88.17	8.87	14.79
4	Baseline model	19.5	76.93	82.07	80.66	22.87	15.81

4.5 The GRM Framework for OGB benchmark

The Omnigenomic Benchmark (OGB) is introduced in 2.7. An RL agent is defined below to find better hyperparameter settings for the model in this benchmark. When a vector of batch size and hyperparameters is selected and training begins, a training log is generated for that vector. Based on this log, the root mean squared error (RMSE) can be calculated as a reward signal. As a lower RMSE is better, we consider its negative value as the reward signal.

- **RL state** : The state is defined by the discrete setting of the
state set = {2, 4, 8, 16}
- **RL Action ($\mathcal{A}$)**: An action is a mapping that selects a unique combination

of values for the remaining, tunable hyperparameters:

$$\mathcal{A} = (\text{"learning_rate"}, \text{"optimizer"}, \text{"scheduler"})$$

The RL-based HPO process is described in Algorithm 6.

Algorithm 6 Training with RL algorithm for OGB Benchmark

- 1: Let $\mathcal{S} = \{2, 4, 8, 16\}$
 - 2: Let $\mathcal{A} = (\text{"learning_rate"}, \text{"optimizer"}, \text{"scheduler"})$
 - 3: Initialize Q-table $Q(s, a)$ to 0 for all states $s \in \mathcal{S}$ and actions $a \in \mathcal{A}$.
 - 4: **for** episode = 1 to MAX_EPISODES **do**
 - 5: Select a batch size state $s \in \mathcal{S}$.
 - 6: Choose action $a \in \mathcal{A}$.
 - 7: Train the model with batch size s and hyperparameters a .
 - 8: Retrieve the training log generated during the training process.
 - 9: Calculate the Root Mean Squared Error (RMSE) from the training log.
 - 10: Calculate the reward signal: $R \leftarrow -\text{RMSE}$.
 - 11: Select the next state s' .
 - 12: Update $Q(s, a)$ using the **Q-Learning** update rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha [R + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$
 - 13: **end for**
 - 14: **return** The optimal hyperparameter policy $\pi^*(s) = \arg \max_a Q(s, a)$.
-

4.6 GRM Framework

The Omnigenomic Benchmark (OGB) benchmark is introduced in 2.7. RL agent can be employed to find better hyperparameter settings for the model in this benchmark. When a vector of batch size and hyperparameters are selected and training begins, a training log is generated for that vector. Based on this log, a score can be calculated as a reward signal. Due to a range of uncertainties, including non-deterministic GPU operations that occur during the training of deep learning algorithms on GPUs, variations in model performance based on hyperparameters, and other sources of uncertainty, this problem inherently contains a significant amount of noise.

An RL agent can be defined for this problem using the problem modeling based on vector from the batch size and hyperparameter and the score as the reward signal. During training for an RL agent, it is possible to verify the conditions

listed in Section 3.2.5. The problem has a non-stationary reward value for the states, which is a vector of the model and its hyperparameter.

Within the GRM framework, the unknown true environment state is conceptually defined as the one encompassing full observability of all noise sources. Since numerous sources of uncertainty remain unobserved—even when the training model and its hyperparameter are considered as a single observable component—a mapping function must be introduced. This function is essential for establishing the GRM framework; it maps the observable state to the batch size used for training. Thus, the new state in the GRM is represented by the batch size itself, derived via this mapping function. The state space and action space in GRM framework for this problem are defined as below:

The problem is the need to select from a variety of models and hyperparameters for training on the OGB benchmark, such as the architectures detailed below.

- **GRM state** : The state is defined by the discrete setting of the state set = {2, 4, 8, 16}
- **GRM Action ($\mathcal{A}$)**: An action is a mapping that selects a unique combination of values for the remaining, tunable hyperparameters:

$$\mathcal{A} = ("learning_rate", "optimizer", "scheduler")$$

The selected batch size and hyperparameters will be used for training. During the training process, a detailed log will be generated for each epoch, capturing various model training parameters. The training log recurrently records two critical performance signals that are used in the final score calculation:

- **f1_score**: This metric is the harmonic mean of precision and recall and is typically used for classification tasks. A higher F_1 score indicates better performance, especially in scenarios with class imbalance.
- **root_mean_squared_error (RMSE)**: This metric is used for regression analysis and quantifies the average magnitude of the prediction errors. Mathematically, it is the standard deviation of the residuals, and a lower RMSE value indicates a better fit to the data.

The formal representation of the score is given by a weighted linear combination, where P_1 is f1_score and P_2 is root_mean_squared_error average values extracted from training information.

$$\text{Score} = w_1 \cdot P_1 + w_2 \cdot P_2$$

The sign of the coefficients w_1 and w_2 determines the direction of the optimization for the performance metrics P'_1 and P'_2 . If the metric should be minimized (lower value is better), w_i must be negative to convert the minimization goal into a reward maximization objective. So for RMSE, a negative constant should be selected as weight. If the metric should be maximized (higher value is better), w_i must be positive to align the metric with the maximization goal. Based on this, A positive value should be selected as a weight for `f1_score`.

GRM Reward function

Based on the defined action and state space the reward function for the GRM will be defined based on the model architecture.

For each state (batch size) $s \in \mathcal{S}$, 10 initial training runs are executed. For each of these 15 runs, a hyperparameter configuration (action) $\lambda \in \Lambda$ is randomly selected. For the k -th run with batch size s , the following data points are collected:

1. Penalty Signals $P_1^{(k)}$ and $P_2^{(k)}$: These are calculated from training log, as defined previously.
2. True Performance $\text{Perf}^{(k)}$: This is the actual efficacy of the model after training, evaluated on test set of samples from the OGB benchmark. A higher value indicates better performance.

For each state s , the collected data (10 pairs of penalty signals and true performance values) are used to calibrate the optimal weights $\mathbf{w}_s = (w_{1,s}, w_{2,s})$. The reward estimator is designed to linearly map the penalty signals to the true performance:

$$\text{Perf} \approx w_{1,s} \cdot P_1 + w_{2,s} \cdot P_2$$

The calibration of $\mathbf{w}_s$ for batch size s is achieved by solving a linear regression problem that minimizes the squared error between the estimated reward and the true performance over the 10 data points. The resulting coefficients $\mathbf{w}_s = (w_{1,s}, w_{2,s})$ are the state-specific penalty weights.

The GRM-based HPO process is formalized in Algorithm 7. This algorithm first learns a batch-size-specific reward model and then uses that model to guide the search for optimal hyperparameters in a subsequent online phase.

Algorithm 7 Training with GRM for OGB Benchmark

Phase 1: Offline Reward Estimator Calibration

- 1: Let $\mathcal{S} = \{2, 4, 8, 16\}$ be the set of states (batch sizes)
- 2: Initialize dataset $D_s \leftarrow \emptyset$ for each state $s \in \mathcal{S}$.
- 3: $\Lambda = (\text{"learning_rate"}, \text{"optimizer"}, \text{"scheduler"})$
- 4: **for** each batch size $s \in \mathcal{S}$ **do**
- 5: **for** $k = 1$ to 10 **do** $\triangleright$ 10 training runs are executed for each state
- 6: Sample a random hyperparameter configuration $\lambda^{(k)} \in \Lambda$.
- 7: Train model with batch size s and hyperparameters $\lambda^{(k)}$
- 8: Calculate Penalty Signals $P_1^{(k)}$ and $P_2^{(k)}$.
- 9: Obtain True Performance $\text{Perf}^{(k)}$ from the held-out set.
- 10: Store penalty vector $\mathbf{P}^{(k)} = (P_1^{(k)}, P_2^{(k)})$.
- 11: $D_s \leftarrow D_s \cup \{(\mathbf{P}^{(k)}, \text{Perf}^{(k)})\}$.
- 12: **end for**
- 13: Perform **Linear Regression** on D_s to find the optimal weight vector $\mathbf{w}_s = (w_{1,s}, w_{2,s})$ by minimizing:

$$\sum_{k=1}^{15} (\text{Perf}^{(k)} - (w_{1,s} \cdot P_1^{(k)} + w_{2,s} \cdot P_2^{(k)}))^2$$

- 14: **end for**

Phase 2: Online Q-Learning-based Hyperparameter Search

- 15: Initialize Q-table $Q(s, \lambda)$ for all $s \in \mathcal{S}$ and $\lambda \in \Lambda$.
 - 16: **for** episode = 1 to MAX_EPISODES **do**
 - 17: Select a batch size state $s \in \mathcal{S}$.
 - 18: Choose action (hyperparameters) λ from $Q(s, \cdot)$.
 - 19: Train model with batch size s and hyperparameters λ for a limited number of epochs.
 - 20: Calculate Partial Penalty Signals P_1^{partial} and P_2^{partial}
 - 21: $\triangleright$ Use the reward estimator to generate the reward
 - 22: $R^{\text{est}} \leftarrow w_{1,s} \cdot P_1^{\text{partial}} + w_{2,s} \cdot P_2^{\text{partial}}$.
 - 23: Update $Q(s, \lambda)$.
 - 24: **end for**
 - 25: **return** The optimal hyperparameter policy $\pi^*(s) = \arg \max_{\lambda} Q(s, \lambda)$.
-

4.6.1 Results

The Weighted Total definition in the OGB and details of the evaluation criteria are described in Section 2.7. Table 4.5 illustrates the Weighted Total evaluation of competing algorithms in the OGB benchmark, comparing the RL algorithm and the GRM framework. A higher Weighted Total determines a better algorithm in the final ranking.

Table 4.5: Comparison of Weighted Total scores between the RL algorithm and GRM framework on the OGB Benchmark

#	Algorithm	Weighted Total ₁
1	GRM	3.68
2	RL	3.45

Table 4.6 illustrates the Weighted Total evaluation of competing algorithms in the ECML 2025 final ranking. The details of the results are mentioned in [Cha25]. The GRM framework achieved the highest performance in the ECML Discovery Challenge on Foundation Models for Genomic Modeling and Understanding by the Weighted Total value of 3.68.

Table 4.6: Comparison of Weighted Total for different frameworks on the OGB Benchmark.

#	Team Name	Weighted Total ₁
1	GRM	3.68
2	Second Rank Framework	3.56
3	Third Rank Framework	3.56

4.7 Conclusion

The Generative Reward Machine was introduced as an extension of the Reward Machine concept to train better models for hard class of Reinforcement Learning problems which are introduced in chapter 3. This core concept was implemented and evaluated across diverse complex domains, including Reconnaissance Blind Chess, a biometric footprint recognition benchmark, and the OGB benchmark. The empirical results from the applications confirm the GRM’s utility in providing a stable and efficient learning signal. While the GRM-trained agent for Reconnaissance Blind Chess was competitive, its peak utility was demonstrated in the HPO tasks, validating the GRM’s capability to transform

environments that violate Markovian assumptions into a structured learning environment suitable for the examples mentioned.

Chapter 5

GymDC environment

5.1 Introduction

The initial step in tackling any reinforcement learning problem is to determine the environment in which the agent will operate. In real-world applications, the agent interacts with the physical or operational environment. In most cases, that is impractical, and so a simulated environment needs to be created.

In a typical RL workflow, the agent learns by interacting with the environment over many episodes, exploring the action space and observing outcomes to develop a policy that maximizes cumulative reward. This interaction process is central to policy learning. Once a policy is learned, we can evaluate its quality by comparing it to baseline policies or competitor algorithms. These comparisons must be performed within the same environment or simulation framework to ensure fairness and consistency.

The simulator, which represents the environment in software, outlines the problem dynamics and facilitates repeated interaction for learning and evaluation. While simulators may abstract away some real-world details, they must explicitly define and implement the core components—state space, action space, and reward function—used by reinforcement learning algorithms.

In this chapter, we describe the Gym-DC environment which simulates a Distribution Center in the Physical Internet. Section 5.2 gives an overview of the simulator. Section 5.3 describes the main parameters required to enable reinforcement learning. Section 5.4 presents the baseline policies implemented in the simulator and compares their performance in different scenario settings.

5.2 Distribution Center Simulator

Within the framework of the Physical Internet, a distribution center (DC) is an open, certified facility adapted to receive, store, and ship containers [MMB13]. In the vision for the Physical Internet, autonomy is widely distributed across various locations and entity types. A single agent making decisions in a distribution center will encounter varying degrees of uncertainty regarding the outcomes of its actions in different scenarios. In any realization of the Physical Internet, simulated environments will be essential for evaluating machine learning systems that infer new policies, as financial penalties could arise from poor decision-making during the exploration phase. At the conceptual level, the Physical Internet is a network of DCs. Packets travel from origins to destinations through at least one DC and perhaps many DCs, with the transport provider for each onward leg selected at each DC. The system can be represented as a graph, with origins, destinations and DCs as vertices. If there is at least one transport provider offering a service between two locations, then these vertices are joined by an edge. The entities of the Physical Internet and the conceptual idea is depicted in Figure 5.1.

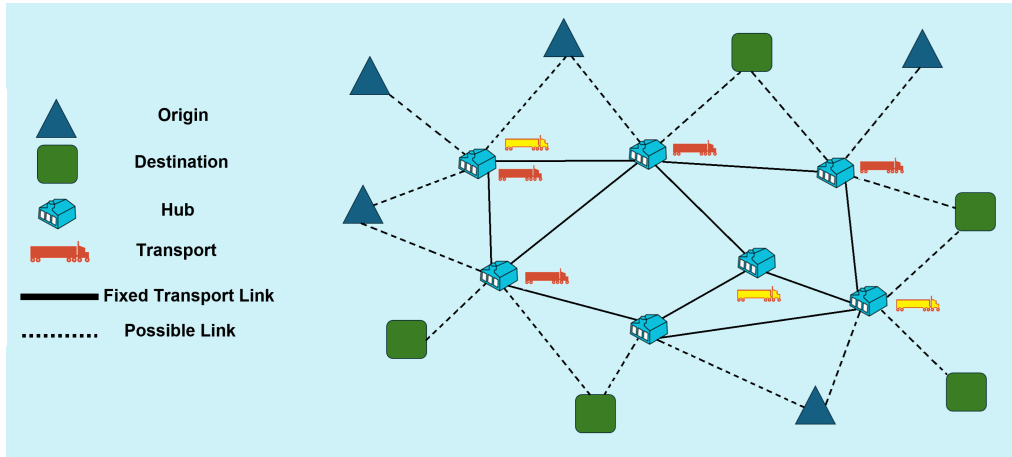


Figure 5.1: Physical Internet Graph

5.2.1 Distribution Center System Design Overview

We present a high-level overview of a Physical Internet distribution center, outlining the internal processing logic through a conceptual simulator. The system, in this context, refers to a single distribution center (hub) within the broader PI network. A packet refers to a physical shipment unit such as a modular container or parcel that moves through the PI infrastructure. These packets act as

the primary input.

Upon arrival at the DC, each packet is first processed in the Entry Zone, where its initial status is assessed. Depending on available resources and policies, a packet may be dropped at this stage. Approved packets proceed either to Storage, where they await further instructions, or directly to the Load Preparation area for bundling and transfer. Packets rejected due to insufficient storage capacity are classified as dropped packets. During performance evaluations, penalties associated with all dropped packets are explicitly accounted for. The abstract form of the PI DC is shown in Figure 5.2.

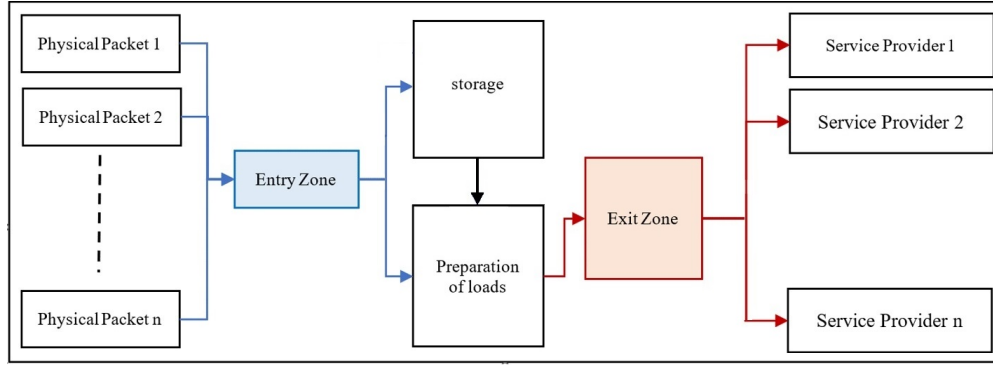


Figure 5.2: Conceptual diagram for DC

5.2.2 Distribution center modeling

The aim of the Distribution Centre is to find the best transportation services for the set of packets it receives over a time period. It is a sequential decision problem, with some decisions being required before all packets are known, and before the results of some previous decisions are known. We assume discrete time steps. The simulation progresses through these time steps in an event-driven manner. This means the clock jumps to the next moment an event is scheduled to occur—whether it is the arrival of a new packet, the scheduled departure of a service provider, or a packet’s deadline expiring. Consequently, the interval between time steps is variable, and many can happen in one day. Time-steps within the system are defined as discrete intervals corresponding to any moment at which a new event occurs. At each step, the DC will receive some packets, and must make a decision on all packets that it receives in that time-step and on all packets previously received but not yet dispatched or dropped. The DC knows the schedule of service providers for the next few days. Each packet is characterized by an id, an ultimate destination, a budget for transportation, its weight, its arrival time at the DC, a deadline for delivery,

and penalties to be incurred when a packet exceeds its due date or is not delivered. Each service provider is characterized the destinations it offers, a price it charges per kg for each destination, an expected delivery time per destination, a departure time, and the number of packets it can accept.

After a packet arrives and is placed into storage, the system, at time-steps, may allocate specific packets to particular service providers. These allocations remain provisional and may be adjusted across multiple time-steps until the departure of a service provider, at which point assignment decisions become final and the packets are loaded onto the designated provider. This allocation of packets to service providers is referred to as "matching" in the context of this process. The capacity of service providers and storage services will be updated after each round of matching. The decision system gains profit based on the difference between the budget of the packet and the service provider charge, and is penalized for dropped or late packets. Following dispatch, the final reward is determined by the delivery outcome. If the service provider delivers a packet by its deadline, the system earns a profit. However, if the packet is delivered after the deadline, a late-delivery penalty is applied. A penalty is incurred for packets that are never delivered. Section 5.2.5 presents the formulae for calculating profit, penalty, and system performance.

Each packet and service provider is defined by a set of key attributes that must be specified within the simulation environment. To model a diverse set of individual packets and service providers, appropriate values must be assigned to these parameters. In some cases, attributes such as size, destination, deadline, budget, late delivery penalty, and non-delivery penalty are not set to fixed values but are instead generated from probability distributions. For these attributes, the distribution's mean and standard deviation need to be specified in advance, allowing values to be sampled accordingly during simulation. A summary of the key parameters are shown in Table 5.1.

Table 5.1: Important simulator parameters

Parameter	Description
num packets	Number of packets in the which are arrived in the simulation period.
num sp	Number of service providers.
num days	Number of days of the period.
num departures sp	Number of departures for every service provider in the simulation period.
num destination	Total number of destinations.
mean packet weight	Mean parameter of weight for packets.
stddev packet weight	Standard deviation of weight for packets.
mean packet budget	Mean parameter of budget for packets.
stddev packet budget	Standard deviation of budget for packets.
mean packet penalty	Mean parameter of penalty.
stddev packet penalty	Standard deviation of penalty.
mean packet time to deadline	Mean parameter of time to deadline from arrival time.
stddev packet time to deadline	Standard deviation of time to deadline from arrival time.
mean service provider price per kg	Mean parameter of price per KG of service provider.
stddev service provider price per kg	Standard deviation of price per KG of service provider.
mean service provider time	Mean parameter of delivery service provider delivery time.
stddev service provider delivery time	Standard deviation of service provider delivery time.
mean service provider capacity	Mean parameter of service provider capacity.
stddev service provider capacity	Standard deviation of service provider capacity.
storage capacity	The value for the storage capacity.
window time	Number of following days which the providers schedule of service are known for the system.

5.2.3 Understanding Incoming Packet Parameters in Simulator Settings

The total number of packets in the simulation is defined by a parameter named `num_packets`. These packets must be distributed across the number of days for which the distribution center is simulated. Accordingly, the system determines the number of packets arriving on the first day, second day, and so forth until the final day of the simulation. The distribution of packets is governed by the parameter `packet_rate`, which specifies the type of distribution applied.

To determine the number of packets allocated per day, five different distribution methods may be employed, as outlined below.

- Uniform: The number of packets per day will be uniform.
- Decreasing: The number of packets per day will decrease in a simulation period.
- Increasing: The number of packets per day will increase in the simulation period.
- Normal: The number of packets per day will have a normal distribution.
- Inverse normal: The number of packets per day will have an inverse normal distribution.
- Fluctuation: The pattern for the number of packets in this option will be selected based on a random variable. When data are generated for a single simulation period and this particular option is activated, the system selects one distribution from among increasing, decreasing, normal, or uniform distributions based on the specified random variable. Consequently, the system might randomly adopt a normal distribution in one execution, while potentially selecting a different distribution in subsequent executions.

Each distribution type used in the simulation is guided by a set of parameters that determine how packets are allocated over time. For the increasing distribution, two main parameters are involved: one sets the minimum number of packets assigned to each day, and the other controls the geometric distribution used to add extra packets. This process leads to a gradually increasing pattern, where later days receive more packets than earlier ones. A similar method is applied to other distribution types as well, with each relying on relevant pa-

rameters to shape the allocation according to the expected behavior of that distribution. Figure 5.3 illustrates an example featuring five distinct distributions spanning 10 days, comprising a cumulative total of 500 packets. Any visual unevenness in the uniform and normal distributions is due to natural sampling variation and the specific y-axis scales, which exaggerate minor fluctuations. Despite this, the underlying distribution parameters are correct.

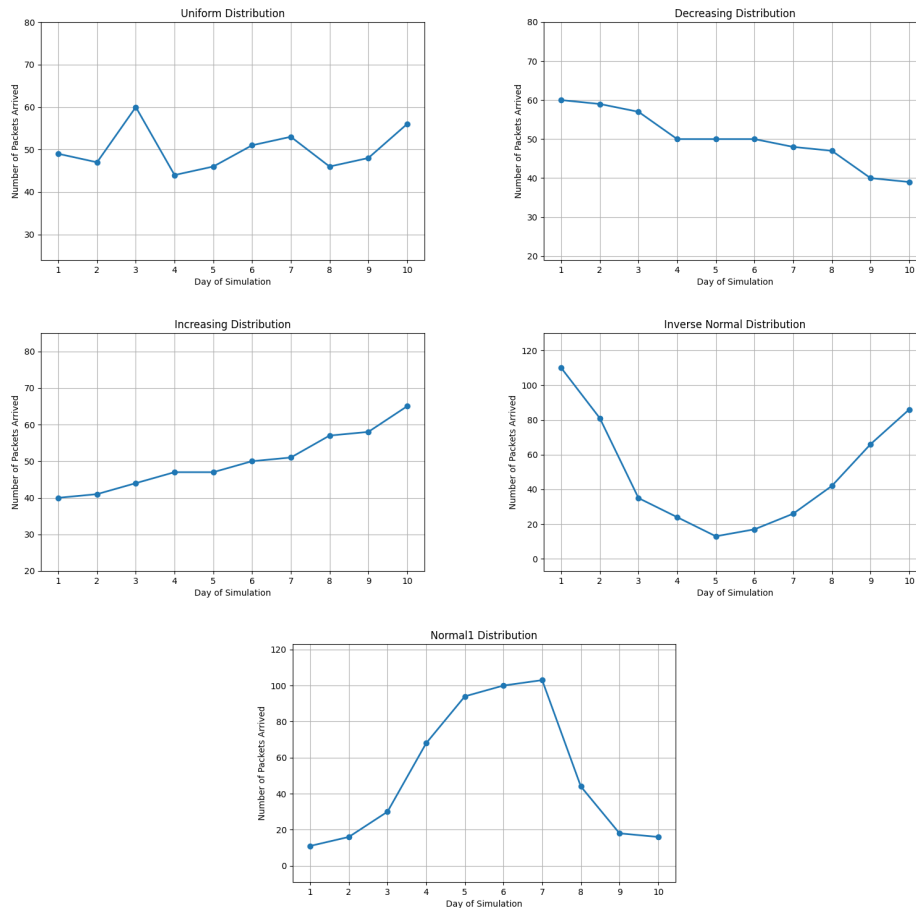


Figure 5.3: Five different distributions over 10 days, with a total of 500 packets.

The flow diagram from the packet's perspective is depicted in Figure 5.4. In the proposed system, if a packet cannot be matched with any currently available service providers, the system conducts a forward-looking assessment. Specifically, it evaluates whether the packet's deadline would expire prior to the arrival of the subsequent batch of service providers. If yes, retaining the packet becomes ineffective. Consequently, the system proactively drops such packets, thereby freeing storage resources and categorizing them among the dropped packets.

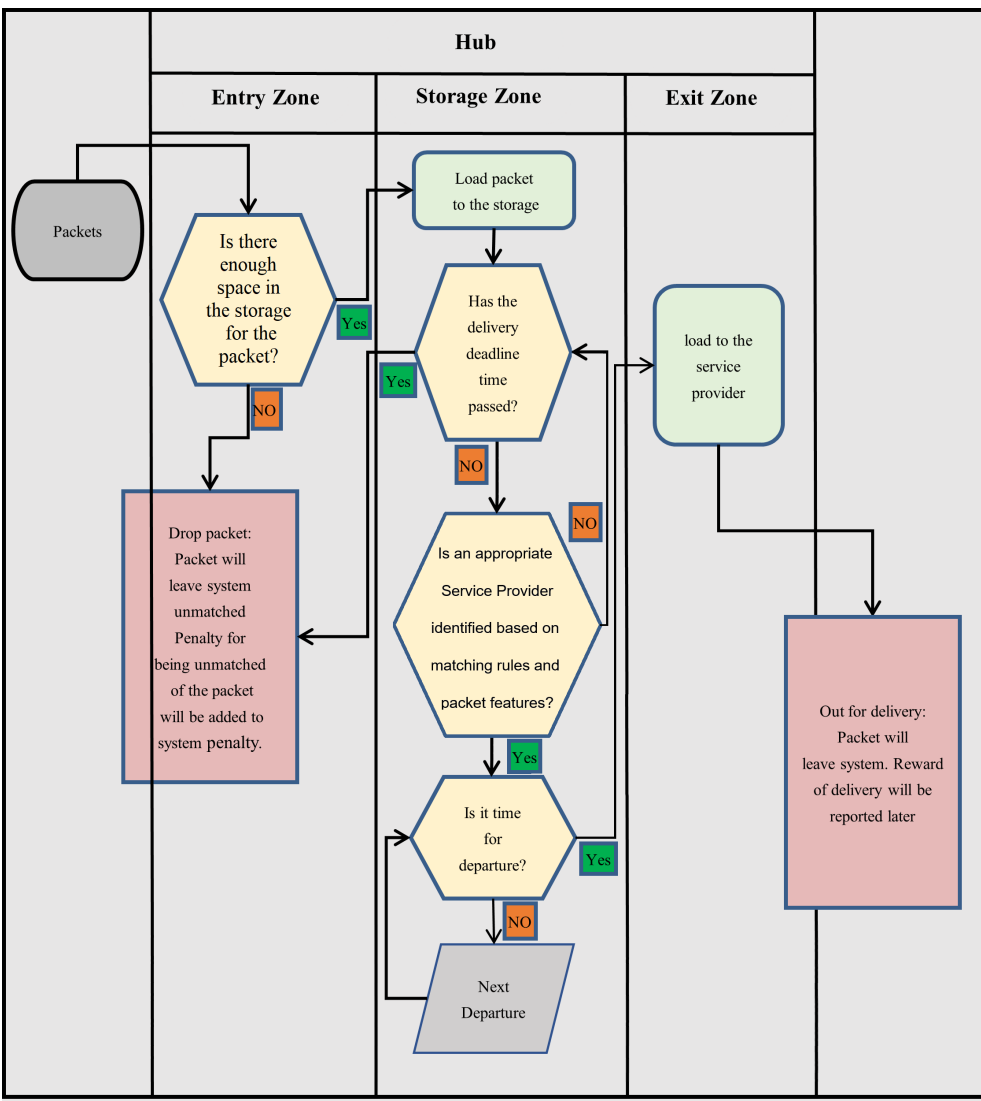


Figure 5.4: Flow diagram from the point of view of the packet

To allocate packets to appropriate service providers efficiently, we introduce a matching mechanism based on predefined hyperparameters and preferences.

5.2.4 Matching Algorithm

Algorithm 8 Matching Algorithm

```

1: Given: A list of packets within their delivery deadline, a list of service
   providers with available capacity, and matching rules.
2: Sort the list of packets by priority based on the matching rules.
3: Sort the list of service providers by preference based on the matching rules.
4: Create an empty list for unmatched packets.
5: for each packet in the sorted list do
6:   Identify an appropriate service provider based on matching rules and
   packet features.
7:   if a matching service provider is found then
8:     Assign the packet to that service provider.
9:     Reduce the available capacity of the service provider.
10:    if the service provider has no more capacity then
11:      Remove the service provider from the list for the next packets.
12:    end if
13:  else
14:    Add the packet to the list of unmatched packets.
15:  end if
16: end for

17: for each packet in the list of unmatched packets do
18:   if the packet will expire before new service providers arrive then
19:     Drop the packet and apply a penalty.
20:   else
21:     Keep the packet in storage for the next matching opportunity.
22:   end if
23: end for

```

The simulator allows for any matching algorithm to be deployed. We provide a parametrised matching system, which separately prioritizes packets and service providers based on specified parameters. It then iterates by selecting the highest priority packet, and allocating it to the highest priority service provider,

until all known service providers are full. As specified in the algorithm 8, for each packet, the system evaluates service provider availability based on capacity and other predefined criteria. Any packets for which the earliest possible delivery would incur a lateness penalty greater than the non-delivery penalty is discarded, and the non-delivery penalty incurred. We provide packet ordering priorities of highest budget, highest penalty, and closest deadline, and service priorities of earliest dispatch, lowest cost, earliest delivery and highest remaining capacity.

5.2.5 Performance Evaluation Under Uncertainty Scenarios

Within the simulator, numerous events occur, such as the arrival of packets, their loading into storage, and the departure of service providers. These events take place within the simulation time configured in the settings. The entire duration of a single simulation execution, from its start to its end, is defined as an episode. It is assumed that the aim of the DC is to maximise its profit minus its penalties over the lifetime of a simulation episode. An episode is one complete run of the simulation, which concludes after a predefined number of time-steps. At the end of each episode, three scenarios define packet delivery outcomes:

- **On-time delivery:** All packets are successfully delivered after being assigned to service providers.
- **Random delay:** In this scenario, 10% of packets are randomly selected and considered late, regardless of the service provider they are assigned to.
- **Unreliable service provider:** Here, 10% of service providers experience delays. Any delivery handled by one of these service providers will be delayed, affecting every packet included in that delivery. If packets expire before these delayed service providers can deliver them, a late penalty is incurred. However, if the packets are still within their expiration time, they are considered successfully delivered. The status of the packets is evaluated during the performance assessment of each episode.

It should be noted that the penalty applied to late or dropped packets is neither a global constant nor a linear function. Instead, each packet is assigned an individual penalty value upon generation, which is drawn from a predefined probability distribution specified within the simulator's configuration param-

ters. At the end of each episode, the simulator computes the profit and penalty for each packet that arrived into the DC. Some packets will remain in the DC, without having been dispatched, but without their deadline expiring. For those packets, a percentage of profit or penalty will be applied based on the number of days to their expiry time. The performance in **Scenario 1** is calculated as follows:

$$Performance = Profit(sent) - Penalty(dropped) - value(remaining)$$

$$Profit = \sum_{i=1}^m [budget(p(i)) - (weight(p(i)) \times Cost(sp))]$$

$$Penalty = \sum_{i=1}^m [penalty(p(i))]$$

In **Scenarios 2 and 3**, a penalty for late packets is applied, and the performance is calculated as:

$$Performance = Profit(sent) - Penalty(dropped) - Penalty(late) - value(remaining)$$

$$Profit = \sum_{i=1}^m [budget(p(i)) - (weight(p(i)) \times Cost(sp))]$$

$$Penalty = \sum_{i=1}^m [penalty(p(i))]$$

5.3 Reinforcement Learning Environment

OpenAI Gym [BCP⁺16] is a widely-used framework designed to facilitate the implementation and reproducibility of reinforcement learning environments. After the final departure, the system will return the performance of the simulation for that period. The fundamental definitions of these sets are as follows:

5.3.1 State Space

Reinforcement Learning assumes that an agent operating within a dynamic system transitions through a sequence of states, where each action taken influences the subsequent state. In Gym-DC, the system state is characterized by two key parameters: *storage capacity value* and *rate*, which are recalculated at each time step.

State Representation

The storage capacity value represents the current available capacity of the system as a fraction of its full capacity. The *rate* parameter is determined based on the system's estimated performance when a greedy policy is applied. At each time step, the worst-case penalty and the best-case profit under a greedy policy

are evaluated. To estimate the penalty, the unmatched penalty of all packets currently stored in the system is summed. For the profit estimation, all available service providers are examined, and the profit from matching each packet to the cheapest suitable provider is calculated.

The computation of the rate parameter depends on the relationship between profit and penalty. If the penalty is greater than or equal to the profit, the rate is determined by the ratio of profit to penalty, with special consideration for cases where profit is zero:

$$\text{rate} = \begin{cases} 0, & \text{if profit} = 0 \\ \frac{\text{profit}}{\text{penalty}}, & \text{otherwise} \end{cases} \quad (5.1)$$

Conversely, if the profit exceeds the penalty, the rate is calculated by adjusting the penalty-to-profit ratio, ensuring it accounts for instances where the penalty is zero:

$$\text{rate} = \begin{cases} 1, & \text{if penalty} = 0 \\ \frac{\text{penalty}}{\text{profit}} + 1, & \text{otherwise} \end{cases} \quad (5.2)$$

State Update

Once these calculations are completed, the system state is updated and returned as a tuple consisting of the storage capacity value and the rate:

$$\text{observation} = [\text{storage capacity value}, \text{rate}] \quad (5.3)$$

5.3.2 Reward Function

Choosing an appropriate reward function is crucial for the agent to learn an optimal policy. The approach taken in this thesis is to consider the performance of each simulation period as a reward. In the Gym-DC reinforcement learning environment, the agent's decision-making process is triggered whenever a service provider departs from the system. At this moment, which constitutes a single timestep, the agent must match packets to the available service providers. Although the agent acts at each timestep, it receives an immediate reward of 0. An episode concludes when the total simulation time has elapsed. Only at the end of the episode is the overall performance calculated and returned to

the agent as a final reward. The value of performance will be mapped to a float value between 0 and 1000 and will return to the agent at the end of each episode.

5.3.3 Action Space

The agent determines the matching policy by selecting suitable matching hyperparameters for the TOPSIS algorithm, which was explained in the section 2.1. Specifically, the action involves choosing the weights used in this algorithm. By configuring these hyperparameters, it is possible to define the ranking priorities for sorting packets and setting matching criteria. For example, if the hyperparameter assigned to packet weight is set to 1, packets will be sorted based on their weights. When multiple packet attributes need to be considered for sorting, the TOPSIS algorithm is used to rank them according to the values of the respective hyperparameters. The same approach is applied to service providers.

5.4 Performance Comparison of Baselines Across Settings

Using the Gym-DC action space, various baselines can be defined based on the different parameters set for the TOPSIS algorithm, as detailed in Section 5.3.3. The baseline policies provided in Gym-DC are:

- **Greedy:** This policy aims to maximize immediate profit at each step. It does this by first selecting the packets that offer the highest budget (potential revenue) and then choosing the service provider that has the lowest transportation cost for the delivery. In this policy, in Algorithm 2, packets are sorted in decreasing order of budget, while providers are sorted in increasing order of cost.
- **fast_departure:** The objective of this policy is to avoid late penalties. To do so, the parameters should be configured to assign the packets with earliest deadline with the service providers with earliest departure time.
- The remaining policies each alternate between the fast_departure policy and the greedy policy based on storage capacity. Greedy is intended for situations where there is an over-supply of providers, and so incurring late penalties should be unlikely. fast_departure is intended for restricted numbers of providers, as waiting for a lower-cost option could risk a late

delivery. When the available storage capacity falls below $K\%$, the system adopts the `fast_departure`; otherwise, it switches to the greedy policy.

We use the following names for four different values of K :

- **Optimistic:** $K = 20\%$ (i.e. will continue to priorities cheapest cost until free storage drops below 20% of total capacity)
- **Stable:** $K = 40\%$
- **Cautious:** $K = 60\%$
- **Conservative:** $K = 80\%$ (i.e. will switch to faster departure as soon as storage is over 20% occupied)

5.5 Simulation settings

The settings 1 to 5 used in the GymDC simulator are listed in table 5.2.

In the first setting, both the service provider capacity and cost are high. Since the incoming packet rate follows a consistent, normal distribution and packets have long deadlines, the risk of incurring late penalties is low. Consequently, the Greedy baseline is expected to perform well. In the second setting, the service provider capacity and cost are set to lower values. This reduction in available capacity increases the risk of late penalties. As a result, the `Fast_departure` and `Conservative` baselines are expected to have better performance. The third setting introduces even greater pressure by combining low provider capacity and cost with short packet deadlines. This configuration makes the risk of late penalties very high. Therefore, the `Fast_departure` baseline is expected to yield the best results. While the first three settings use a normal distribution for incoming packets, settings 4 and 5 introduce a fluctuating packet rate. This variability creates a much more dynamic and unpredictable environment and it is expected that no single baseline will consistently outperform the others. Instead, the top performance will likely be distributed among all the different baselines.

Table 5.2: Comparison of Parameters across Different settings

Parameter	Setting 1	Setting 2	Setting 3	Setting 4	Setting 5
num_packets	500	500	500	500	500
num_sp	5	5	5	5	5
num_days	10	10	10	10	10
num_departures_sp	20	20	20	20	20
num_destination	5	5	5	5	5
mean_packet_weight	10	10	10	15	15
stddev_packet_weight	2	2	2	2	2
mean_packet_budget	30	30	30	15	15
stddev_packet_budget	3	3	3	2	2
mean_packet_penalty_late	6	6	6	6	6
stddev_packet_penalty_late	2	2	2	2	2
mean_packet_penalty_unmatched	100	100	100	80	80
stddev_packet_penalty_unmatched	1	1	1	1	1
mean_packet_time_to_deadline	10	10	5	4	4
stddev_packet_time_to_deadline	3	3	3	3	3
mean_service_provider_price_per_kg	40	20	20	16	16
stddev_service_provider_price_per_kg	2	2	2	2	2
mean_service_provider_delivery_time	5	5	5	2	4
stddev_service_provider_delivery_time	2	2	2	1	1
mean_service_provider_capacity	2000	100	100	50	100
stddev_service_provider_capacity	15	15	15	15	15
mean_storage_capacity	3000	6000	10000	1300	1000
stddev_storage_capacity	50	50	50	100	100
loading_to_storage_time	2	2	2	2	2
window_time_for_service_providers	7	7	7	7	7
packet_rate	Normal	Normal	Normal	fluctuation	fluctuation

5.6 Performance Analysis for Simulation Settings in different scenarios

This section presents a performance analysis for Simulation Settings 1 to 5 under three distinct scenarios: "On-time Delivery", "Random Delay", and "Unreliable Service Provider". To facilitate a clearer comparison of results across different heuristics, we evaluated the performance using 5000 different seeds for the random generator in the simulator. To improve readability and comparability, we applied a moving average technique with a window size of 20. By grouping the data this way, the 5000 seeds are reduced to 250 points on the plot. This method smooths the performance data by averaging values over the specified window size. To generate the plots, these 250 points are sorted based on the baseline with the highest average value. This sorting causes the curve for that specific baseline to increase smoothly, which was a deliberate choice to make the overall trends easier to see. Additionally, the plots show fluctuations that are not visible in the summary tables. They illustrate that even if a baseline has the highest overall average, other baselines can still have higher values in specific ranges.

For each setting and scenario, we include two tables and one plot. The first table reports the pairwise relative performance of different baselines across 5000 simulation runs for a given setting and scenario. Each cell $[i][j]$ represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j . The second table presents the number of seeds for which the row baseline strictly outperforms the column baseline. This table is shown specifically for Setting 1 under the "On-time Delivery" scenario. Cells represent the number of seeds with higher values for the row baseline. Following the two tables, we provide a plot that visually compares the performance of the baselines. Appendix A presents a table for each setting within each scenario. It displays the Selective Performance Comparison Matrix, where each cell $[i][j]$ represents the ratio of the mean performance value in cases where baseline i outperforms baseline j across 5000 simulation runs. This approach highlights instances where strategy i demonstrates superior performance over strategy j at corresponding indices.

In Appendix 8.1, another table for each setting displays the Selective Performance Comparison Matrix, where each cell $[i][j]$ represents the ratio of the mean performance value in cases where baseline i outperforms baseline j across

5000 simulation runs. This approach highlights instances where strategy i demonstrates superior performance over strategy j at corresponding indices. The caption of each table specifies the simulation setting and scenario. While all tables and plots provide information to compare the baselines with each other, the main discussion and analysis of these results are presented in Section 5.7.

Performance Analysis for Simulation Setting 1 in On-time delivery

Table 5.3: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 1, On-time delivery.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	100.00	92.43	78.83	67.09	62.71	61.77
Conservative		100.00	85.28	72.59	67.85	66.82
Cautious			100.00	85.12	79.56	78.36
Stable				100.00	93.47	92.06
Optimistic					100.00	98.49
Greedy						100.00

Table 5.4: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 1 in On-time delivery.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	35	9	0	0	0
Conservative	4620	0	93	2	0	0
Cautious	4690	4593	0	42	4	0
Stable	4740	4739	4647	0	38	0
Optimistic	4806	4806	4749	3229	0	0
Greedy	4827	4827	4775	3292	691	0

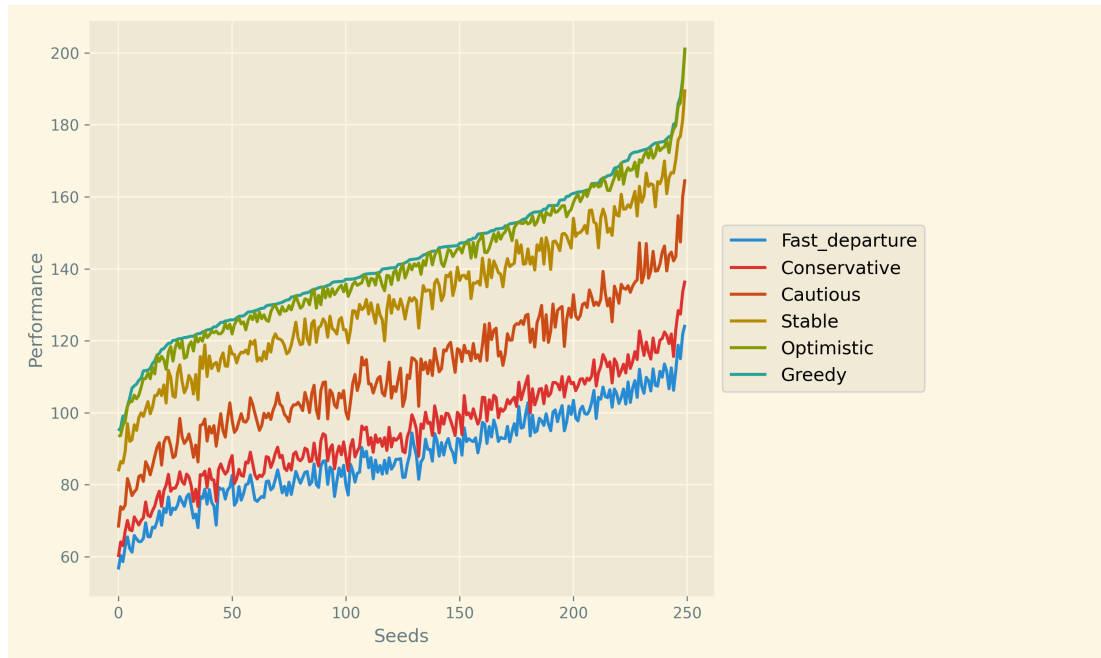


Figure 5.5: Plotting performance of different baselines for Setting 1 in On-time delivery window size = 20

Performance Analysis for Simulation Setting 2 in On-time delivery

Table 5.5: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 2, On-time delivery.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	100.00	97.56	99.61	99.83	99.83	99.83
Conservative		100.00	102.10	102.33	102.33	102.33
Cautious			100.00	100.23	100.23	100.23
Stable				100.00	100.00	100.00
Optimistic					100.00	100.00
Greedy						100.00

Table 5.6: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 2 in On-time delivery.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	100	2116	2297	2297	2297
Conservative	4900	0	4681	4754	4754	4754
Cautious	2884	296	0	1107	1107	1107
Stable	2703	223	176	0	0	0
Optimistic	2703	223	176	0	0	0
Greedy	2703	223	176	0	0	0

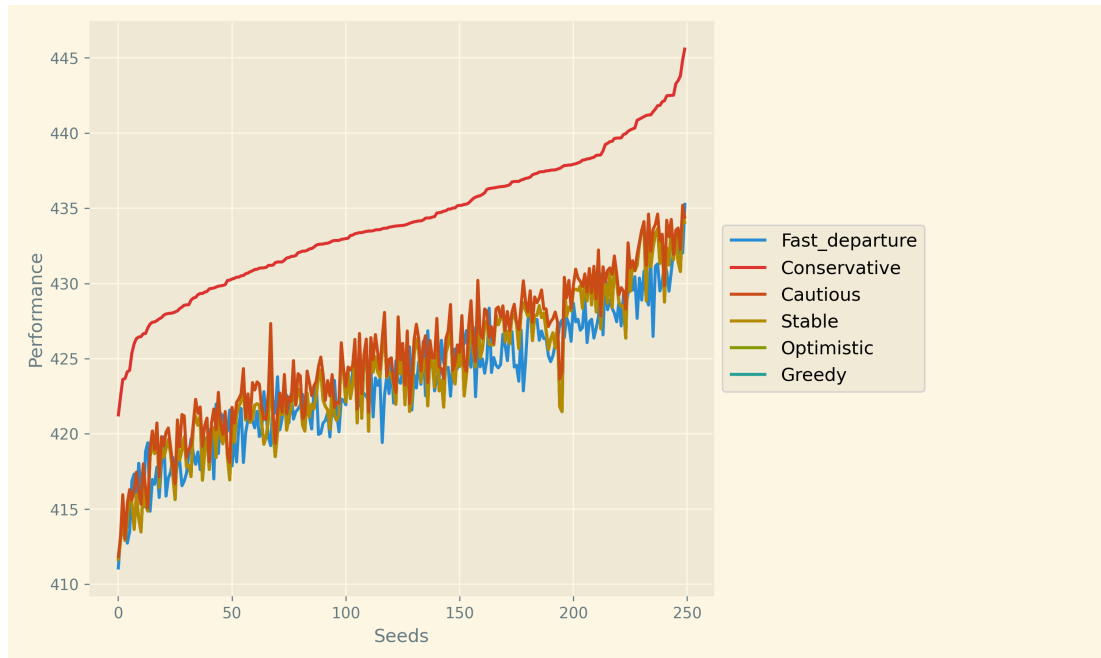


Figure 5.6: Plotting performance of different baselines for Setting 2 in On-time delivery window size = 20

Performance Analysis for Simulation Setting 3 in On-time delivery

Table 5.7: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 3, On-time delivery.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	100.00	107.66	107.66	107.66	107.66	107.66
Conservative		100.00	100.00	100.00	100.00	100.00
Cautious			100.00	100.00	100.00	100.00
Stable				100.00	100.00	100.00
Optimistic					100.00	100.00
Greedy						100.00

Table 5.8: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 3 in On-time delivery.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	4872	4872	4872	4872	4872
Conservative	128	0	5	5	5	5
Cautious	128	2	0	0	0	0
Stable	128	2	0	0	0	0
Optimistic	128	2	0	0	0	0
Greedy	128	2	0	0	0	0

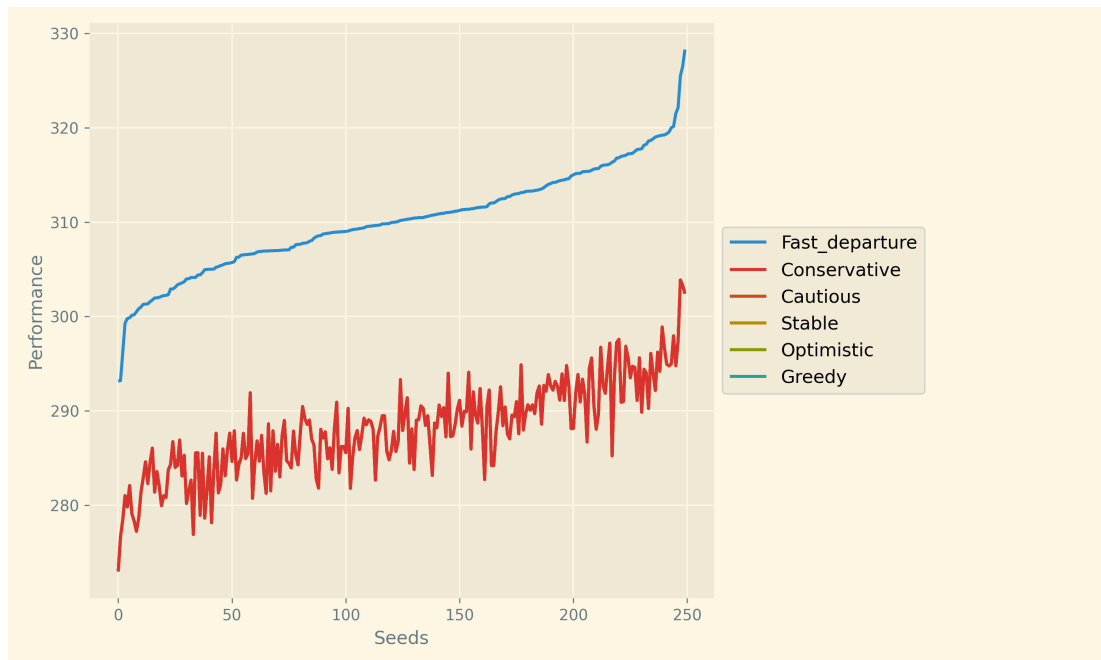


Figure 5.7: Plotting performance of different baselines for Setting 3 in On-time delivery window size = 20

Performance Analysis for Simulation Setting 4 in On-time delivery

Table 5.9: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 4, On-time delivery.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	100.00	100.41	101.95	103.08	101.72	98.71
Conservative		100.00	101.53	102.66	101.31	98.31
Cautious			100.00	101.11	99.78	96.82
Stable				100.00	98.68	95.76
Optimistic					100.00	97.04
Greedy						100.00

Table 5.10: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 4 in On-time delivery.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	2940	4238	4258	3296	1849
Conservative	1439	0	4120	4190	3160	1701
Cautious	736	813	0	3441	2322	1106
Stable	742	810	1558	0	1469	656
Optimistic	1704	1840	2678	3530	0	724
Greedy	3151	3299	3894	4344	4215	0

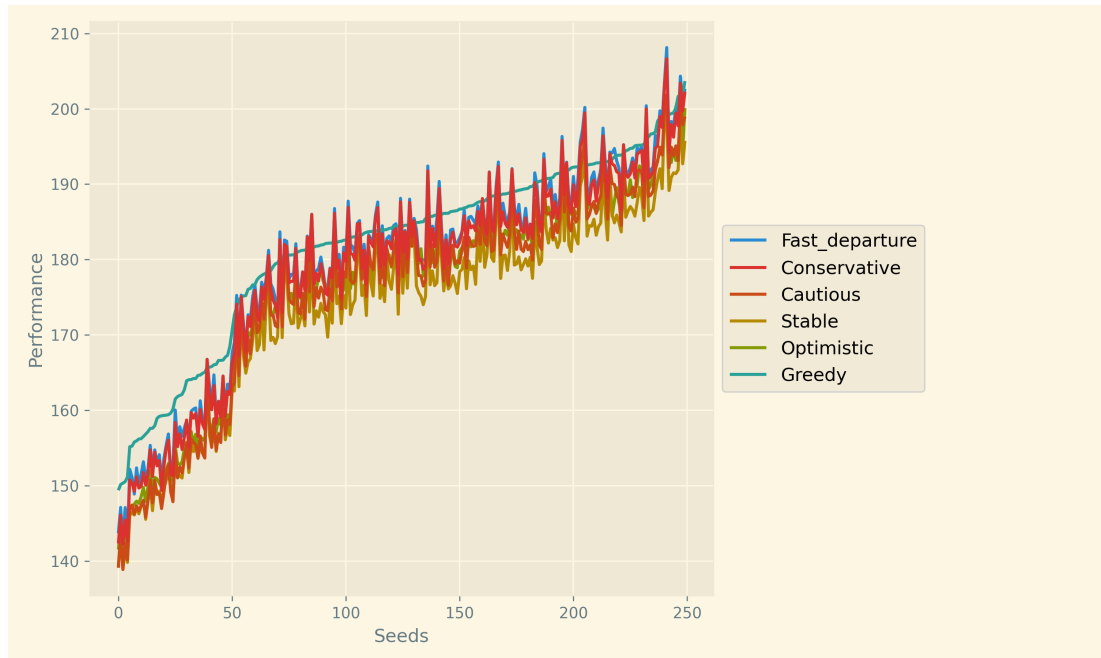


Figure 5.8: Plotting performance of different baselines for Setting 4 in On-time delivery window size = 20

Performance Analysis for Simulation Setting 5 in On-time delivery

Table 5.11: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 5, On-time delivery.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	100.00	98.03	94.01	92.31	92.06	92.69
Conservative		100.00	95.90	94.17	93.91	94.56
Cautious			100.00	98.19	97.93	98.60
Stable				100.00	99.73	100.42
Optimistic					100.00	100.69
Greedy						100.00

Table 5.12: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 5 in On-time delivery.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	257	93	152	317	573
Conservative	4728	0	242	335	546	848
Cautious	4907	4758	0	1090	1352	1712
Stable	4848	4665	3909	0	2160	2529
Optimistic	4683	4454	3646	2793	0	2568
Greedy	4427	4152	3288	2432	1752	0

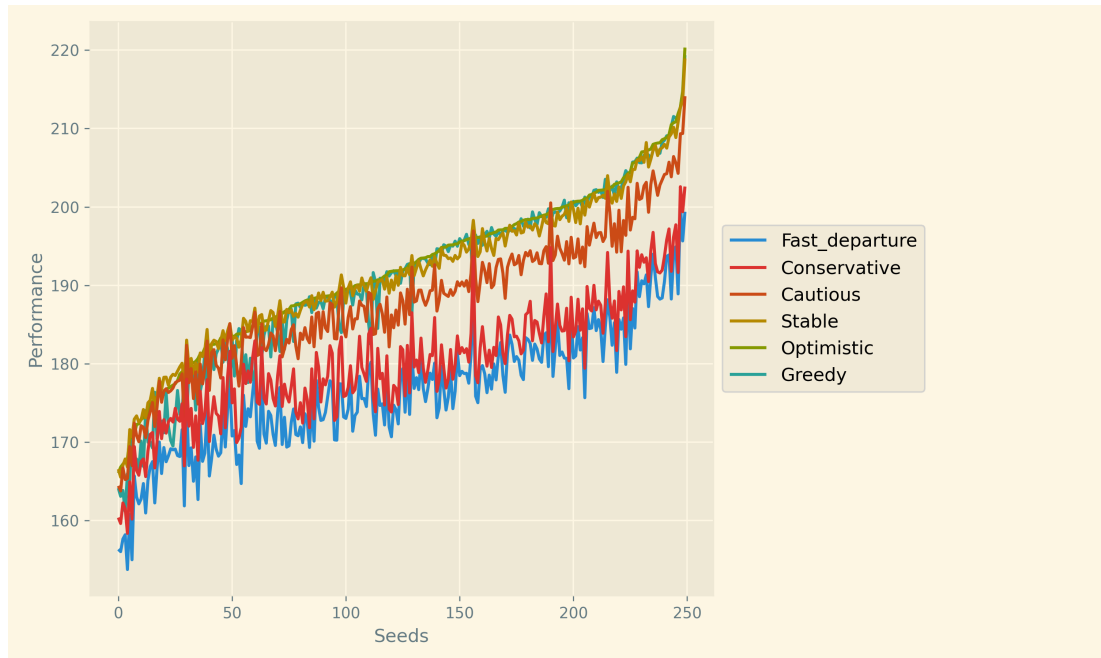


Figure 5.9: Plotting performance of different baselines for Setting 5 in On-time delivery window size = 20

Performance Analysis for Simulation Setting 1 in Random delay

Table 5.13: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 1, Random delay.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	100.00	94.81	84.76	75.29	71.50	70.65
Conservative		100.00	89.40	79.41	75.42	74.52
Cautious			100.00	88.83	84.36	83.36
Stable				100.00	94.97	93.84
Optimistic					100.00	98.81
Greedy						100.00

Table 5.14: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 1 in Random delay.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	1051	322	56	14	11
Conservative	3811	0	580	102	26	16
Cautious	4548	4291	0	263	60	42
Stable	4834	4787	4612	0	379	303
Optimistic	4906	4894	4844	3304	0	126
Greedy	4930	4925	4882	3401	807	0

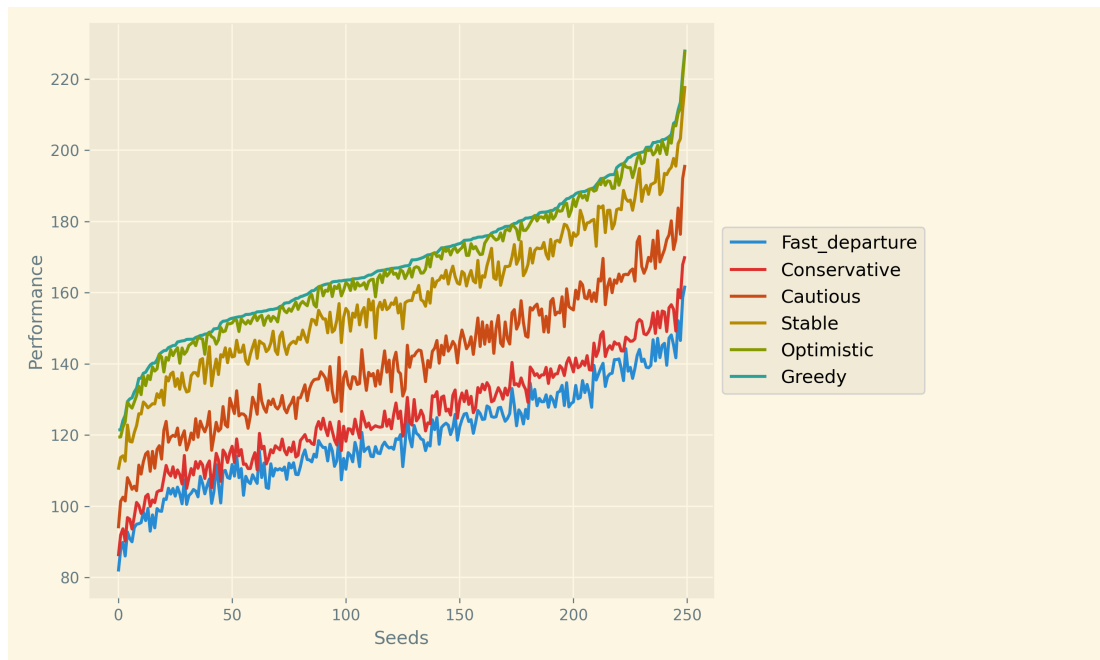


Figure 5.10: Plotting performance of different baselines for Setting 1 in Random delay window size = 20

Performance Analysis for Simulation Setting 2 in Random delay

Table 5.15: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 2, Random delay.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	100.00	97.66	99.74	99.97	99.97	99.97
Conservative		100.00	102.13	102.37	102.37	102.37
Cautious			100.00	100.23	100.23	100.23
Stable				100.00	100.00	100.00
Optimistic					100.00	100.00
Greedy						100.00

Table 5.16: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 2 in Random delay.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	342	2227	2414	2414	2414
Conservative	4658	0	4428	4532	4532	4532
Cautious	2773	550	0	952	952	952
Stable	2586	446	331	0	0	0
Optimistic	2586	446	331	0	0	0
Greedy	2586	446	331	0	0	0

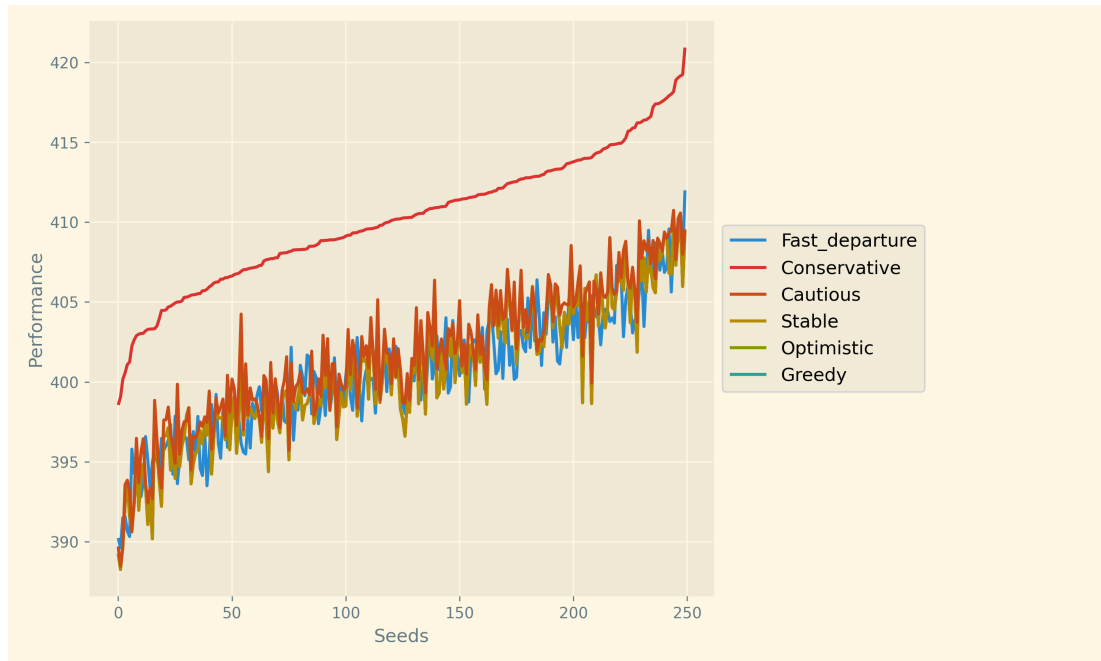


Figure 5.11: Plotting performance of different baselines for Setting 2 in Random delay window size = 20

Performance Analysis for Simulation Setting 3 in Random delay

Table 5.17: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 3, Random delay.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	100.00	107.96	107.96	107.96	107.96	107.96
Conservative		100.00	100.00	100.00	100.00	100.00
Cautious			100.00	100.00	100.00	100.00
Stable				100.00	100.00	100.00
Optimistic					100.00	100.00
Greedy						100.00

Table 5.18: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 3 in Random delay.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	4871	4871	4871	4871	4871
Conservative	129	0	5	5	5	5
Cautious	129	2	0	0	0	0
Stable	129	2	0	0	0	0
Optimistic	129	2	0	0	0	0
Greedy	129	2	0	0	0	0

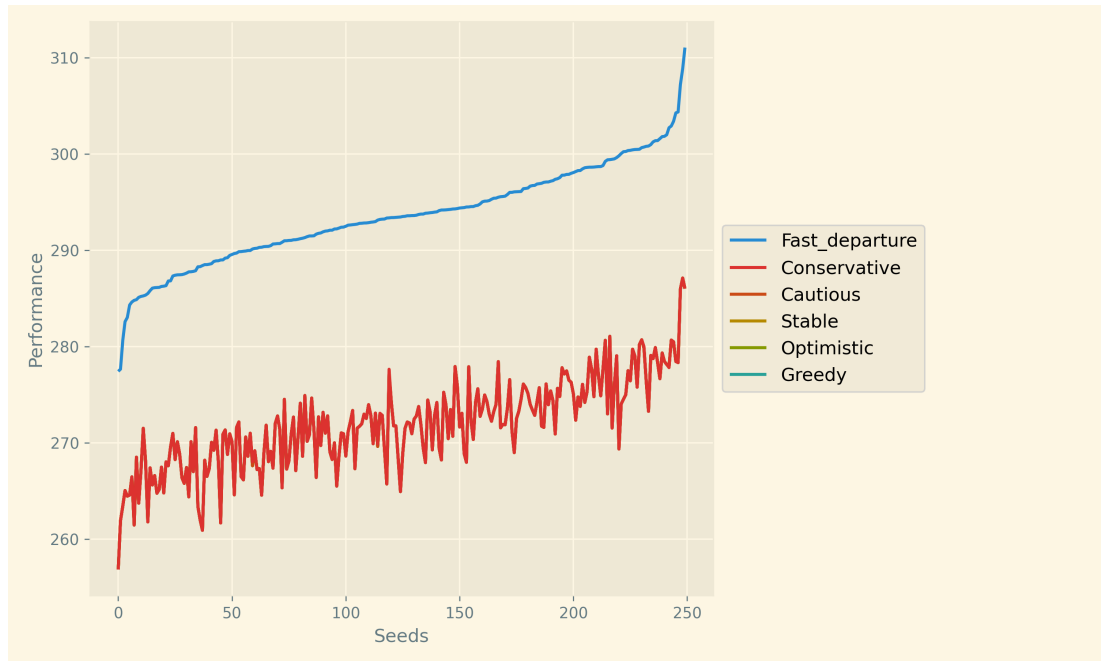


Figure 5.12: Plotting performance of different baselines for Setting 3 in Random delay window size = 20

Performance Analysis for Simulation Setting 4 in Random delay

Table 5.19: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 4, Random delay.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	100.00	100.47	102.24	103.81	103.13	100.88
Conservative		100.00	101.76	103.33	102.65	100.41
Cautious			100.00	101.54	100.87	98.67
Stable				100.00	99.35	97.18
Optimistic					100.00	97.82
Greedy						100.00

Table 5.20: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 4 in Random delay.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	2739	4187	4473	3909	2756
Conservative	1670	0	4001	4358	3780	2549
Cautious	787	936	0	3636	2958	1819
Stable	527	642	1363	0	1977	1087
Optimistic	1091	1220	2042	3022	0	1140
Greedy	2244	2451	3181	3913	3801	0

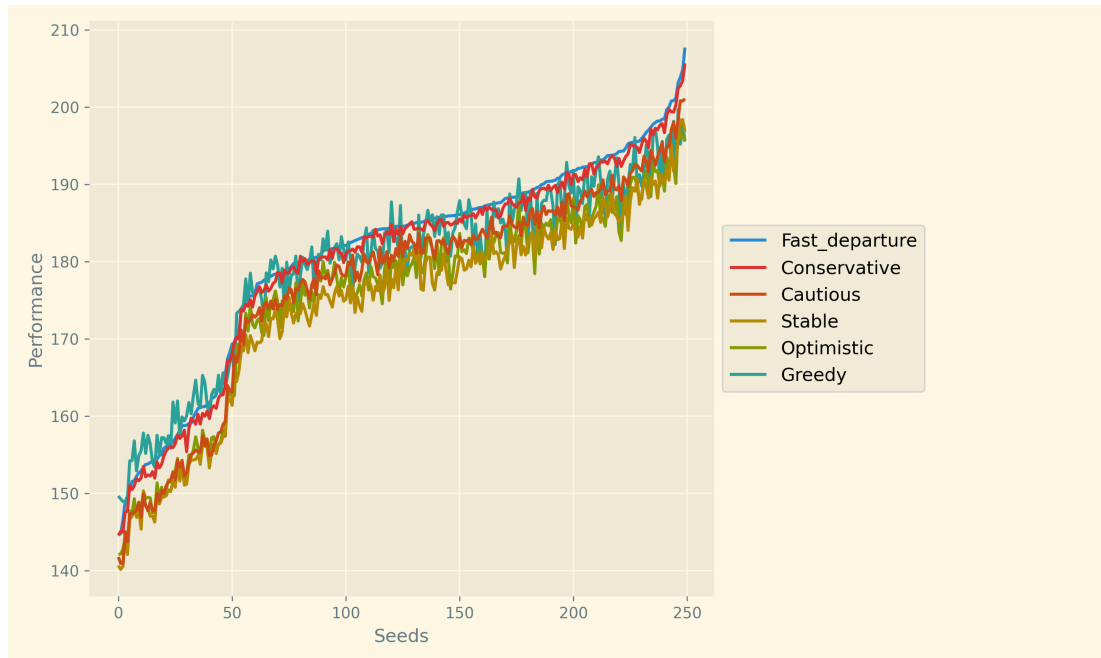


Figure 5.13: Plotting performance of different baselines for Setting 4 in Random delay window size = 20

Performance Analysis for Simulation Setting 5 in Random delay

Table 5.21: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 5, Random delay.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	100.00	98.25	94.85	93.67	93.78	94.64
Conservative		100.00	96.54	95.34	95.46	96.33
Cautious			100.00	98.75	98.87	99.77
Stable				100.00	100.12	101.03
Optimistic					100.00	100.91
Greedy						100.00

Table 5.22: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 5 in Random delay.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	712	219	324	553	908
Conservative	4278	0	505	584	870	1227
Cautious	4781	4493	0	1589	1874	2225
Stable	4676	4415	3409	0	2571	2950
Optimistic	4446	4130	3125	2393	0	2727
Greedy	4092	3773	2773	2017	1599	0

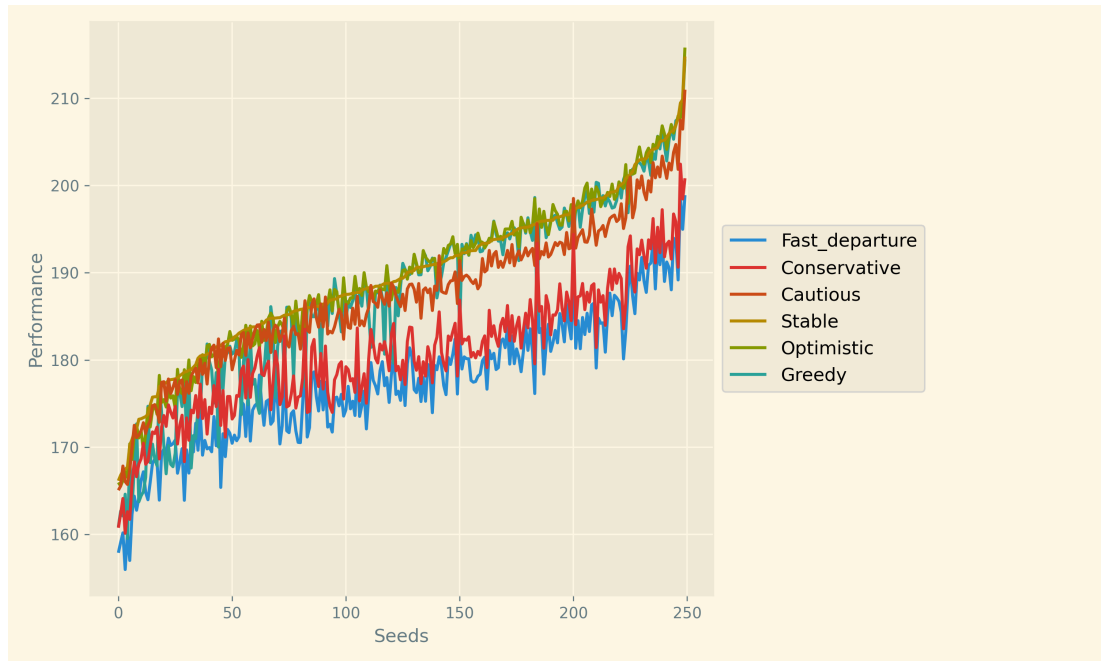


Figure 5.14: Plotting performance of different baselines for Setting 5 in Random delay window size = 20

Performance Analysis for Simulation Setting 1 in Unreliable service provider

Table 5.23: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 1, Unreliable service provider.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	100.00	93.85	81.94	71.12	66.97	66.13
Conservative		100.00	87.31	75.78	71.36	70.47
Cautious			100.00	86.80	81.73	80.71
Stable				100.00	94.16	92.99
Optimistic					100.00	98.76
Greedy						100.00

Table 5.24: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 1 in Unreliable service provider.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	424	301	177	141	150
Conservative	4281	0	354	157	135	137
Cautious	4446	4377	0	190	129	128
Stable	4597	4617	4527	0	172	135
Optimistic	4689	4695	4645	3084	0	36
Greedy	4697	4710	4664	3142	646	0

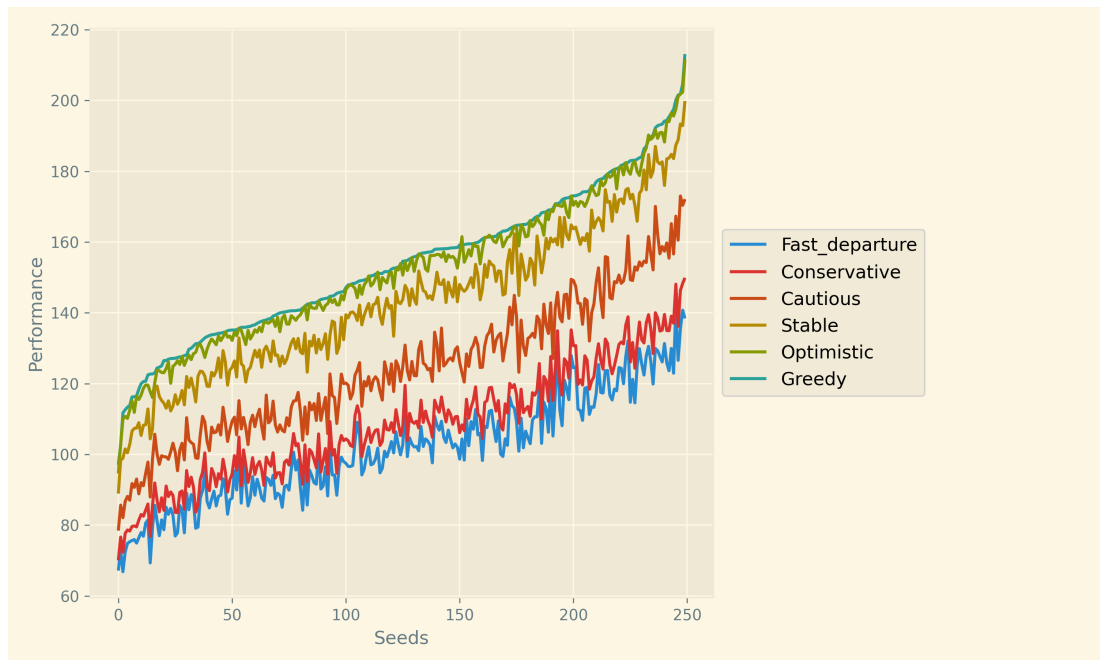


Figure 5.15: Plotting performance of different baselines for Setting 1 in Unreliable service provider window size = 20

Performance Analysis for Simulation Setting 2 in Unreliable service provider

Table 5.25: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 2, Unreliable service provider.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	100.00	97.57	99.61	99.83	99.83	99.83
Conservative		100.00	102.10	102.32	102.32	102.32
Cautious			100.00	100.22	100.22	100.22
Stable				100.00	100.00	100.00
Optimistic					100.00	100.00
Greedy						100.00

Table 5.26: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 2 in Unreliable service provider.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	408	2167	2318	2318	2318
Conservative	4592	0	4480	4558	4558	4558
Cautious	2833	496	0	1071	1071	1071
Stable	2681	418	220	0	0	0
Optimistic	2681	418	220	0	0	0
Greedy	2681	418	220	0	0	0

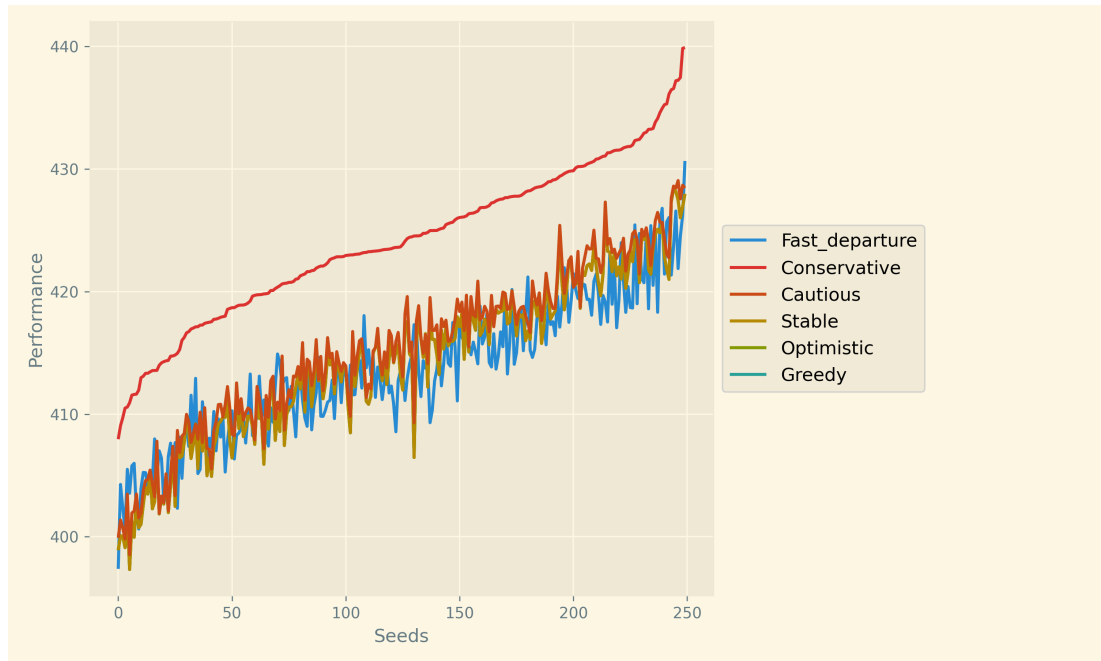


Figure 5.16: Plotting performance of different baselines for Setting 2 in Unreliable service provider window size = 20

Performance Analysis for Simulation Setting 3 in Unreliable service provider

Table 5.27: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 3, Unreliable service provider.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	100.00	107.69	107.69	107.69	107.69	107.69
Conservative		100.00	100.00	100.00	100.00	100.00
Cautious			100.00	100.00	100.00	100.00
Stable				100.00	100.00	100.00
Optimistic					100.00	100.00
Greedy						100.00

Table 5.28: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 3 in Unreliable service provider.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	4659	4659	4659	4659	4659
Conservative	340	0	5	5	5	5
Cautious	340	2	0	0	0	0
Stable	340	2	0	0	0	0
Optimistic	340	2	0	0	0	0
Greedy	340	2	0	0	0	0

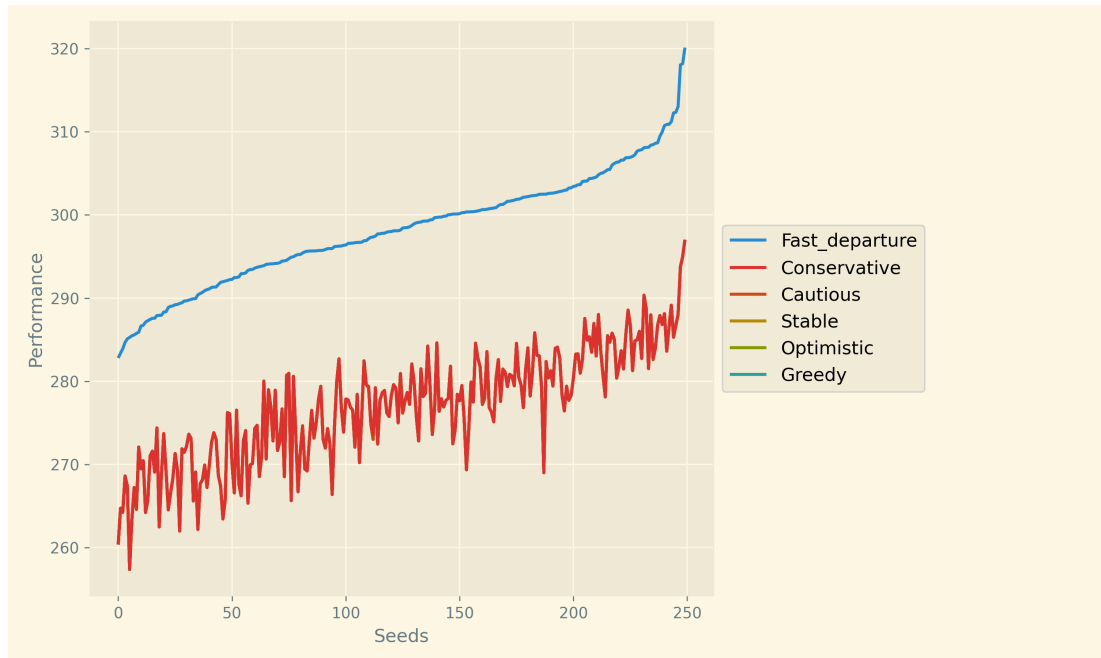


Figure 5.17: Plotting performance of different baselines for Setting 3 in Unreliable service provider window size = 20

Performance Analysis for Simulation Setting 4 in Unreliable service provider

Table 5.29: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 4, Unreliable service provider.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	100.00	100.46	102.12	103.49	102.48	99.93
Conservative		100.00	101.66	103.01	102.01	99.47
Cautious			100.00	101.34	100.35	97.85
Stable				100.00	99.03	96.56
Optimistic					100.00	97.51
Greedy						100.00

Table 5.30: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 4 in Unreliable service provider.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	2978	4320	4364	3608	2330
Conservative	1394	0	4176	4284	3456	2157
Cautious	657	756	0	3599	2648	1543
Stable	636	716	1400	0	1721	960
Optimistic	1392	1544	2351	3278	0	976
Greedy	2670	2843	3457	4040	3951	0

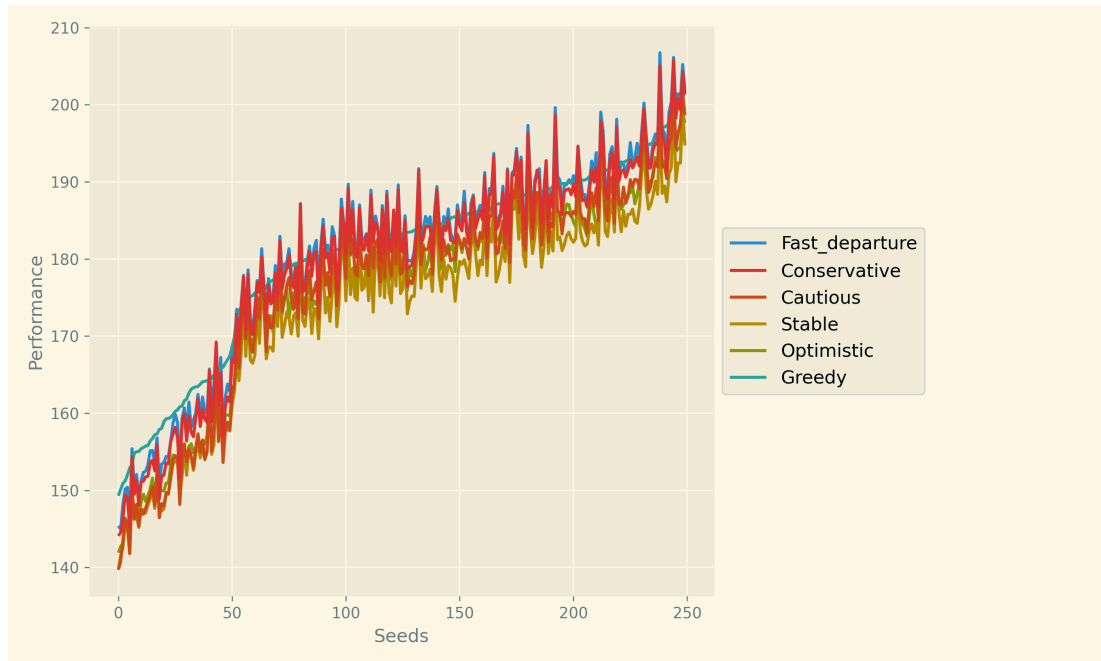


Figure 5.18: Plotting performance of different baselines for Setting 4 in Unreliable service provider window size = 20

Performance Analysis for Simulation Setting 5 in Unreliable service provider

Table 5.31: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 5, Unreliable service provider.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	100.00	98.16	94.58	93.14	93.12	93.91
Conservative		100.00	96.35	94.88	94.87	95.67
Cautious			100.00	98.47	98.46	99.29
Stable				100.00	99.98	100.83
Optimistic					100.00	100.85
Greedy						100.00

Table 5.32: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 5 in Unreliable service provider.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	382	215	351	574	869
Conservative	4603	0	404	577	841	1175
Cautious	4785	4595	0	1358	1665	2033
Stable	4649	4423	3639	0	2418	2771
Optimistic	4426	4159	3333	2540	0	2739
Greedy	4131	3824	2967	2187	1573	0

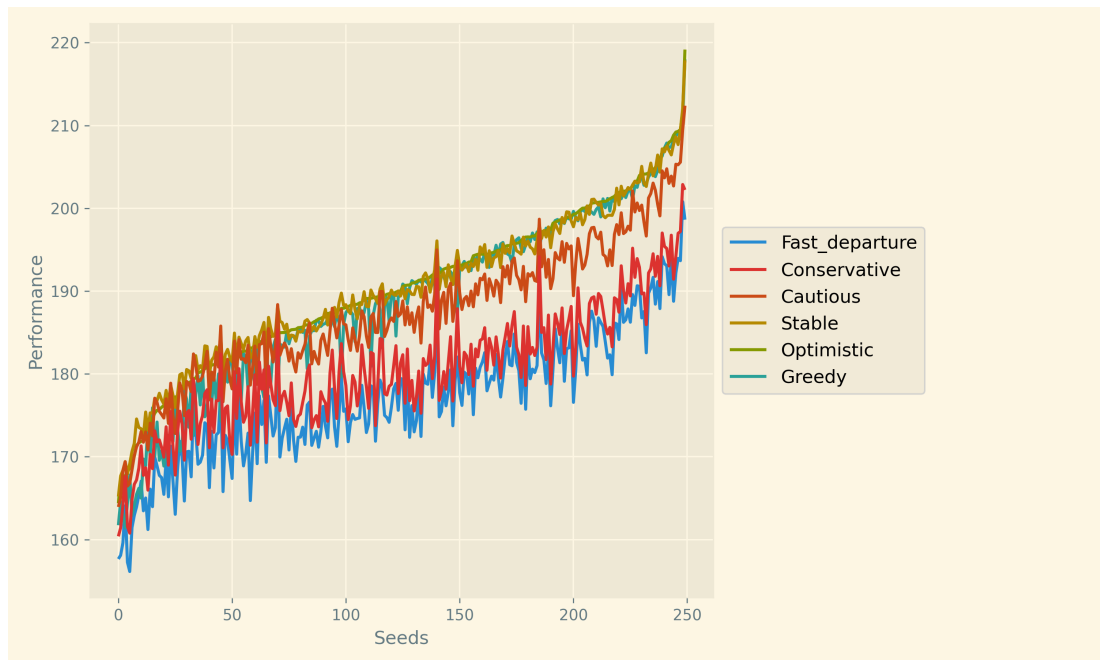


Figure 5.19: Plotting performance of different baselines for Setting 5 in Unreliable service provider window size = 20

5.7 Discussion

This section analyzed the performance of six baseline heuristics—Fast-departure, Conservative, Cautious, Stable, Optimistic, and Greedy—across five distinct simulation settings and three delivery scenarios. A clear distinction in performance emerges when comparing settings which their packet arrival rate is "Normal Distribution" (Settings 1, 2, and 3) against those which their packet arrival rate is "fluctuation" (Settings 4 and 5). In Setting 1, the Greedy and Optimistic strategies achieved the highest performance. Their success can be attributed to their inherent design to maximize immediate opportunities, a strategy that thrives when the flow of tasks is predictable. Similarly, the Conservative baseline was the top performer in Setting 2, indicating that in environments with specific constraints, a more cautious resource allocation strategy yields the best results. In Setting 3, the Fast-departure baseline outperformed others, suggesting that prioritizing rapid processing is most effective under those particular stable conditions. In each of these settings, the leading baseline achieved the highest performance in over 90% of the simulation seeds. Conversely, in Settings 4 and 5, no single baseline consistently outperformed the others. The performance metrics were split among the different heuristics, with various baselines achieving the highest scores across different simulation runs. The fluctuation of incoming packet rates create a complex and dynamic decision-making landscape where the optimal action changes frequently. As a result, a single heuristic is unable to adapt effectively to the shifting demands, leading to a distributed and less predictable performance outcome across the board. The distribution of top-performing baselines is clearly summarized in Table 5.33, which shows the frequency with which each baseline achieved the maximum performance value out of 5000 runs. For the "On-time delivery" scenario, in Setting 1, the Optimistic (4309 times) and Greedy (5000 times, often jointly with Optimistic) baselines dominate. In Setting 2, the Conservative baseline is the clear winner, achieving the top score 4604 times. In Setting 3, Fast-departure achieves the maximum performance in 4872 runs. However, for Settings 4 and 5, the counts are spread more evenly. For instance, in Setting 4, Greedy (2885), Fast-departure (1395), and Conservative (594) all frequently achieve the highest score.

Table 5.33: Frequency of Baselines Achieving Maximum Performance. The table displays the number of times each baseline achieved the highest performance value out of 5000 runs for each scenario and setting. Note that multiple baselines can simultaneously achieve the maximum performance in a single run.

Scenario and Setting	Fast_departure	Greedy	Optimistic	Stable	Cautious	Conservative
On-time delivery						
Setting 1	173	5000	4309	1708	225	173
Setting 2	98	183	183	183	291	4604
Setting 3	4872	128	128	128	128	128
Setting 4	1395	2885	253	45	90	594
Setting 5	12	1636	1515	1367	714	118
Random delay						
Setting 1	66	4529	3855	1564	110	70
Setting 2	322	379	379	379	500	4127
Setting 3	4871	129	129	129	129	129
Setting 4	1844	1902	235	72	220	1069
Setting 5	36	1219	1356	1509	962	219
Unreliable service provider						
Setting 1	229	4738	4137	1759	244	187
Setting 2	398	330	330	330	457	4121
Setting 3	4660	341	341	341	341	341
Setting 4	1729	2433	236	53	119	724
Setting 5	58	1391	1424	1404	855	218

5.8 Deep Reinforcement Learning on Gym-DC

A Reinforcement Learning RL algorithm is used to perform training. After training, the result is a policy, which the RL agent then uses to select actions within its environment. In Section 5.9, the results of the RL agent for the five settings detailed in table 5.2, evaluated across all three defined scenarios: "On-time delivery," "Random delay," and "Unreliable service provider". The details of the algorithms used to train the RL agent are listed in this chapter. In Section 5.8.1, the performance of an RL agent in the Cart-Pole Swing-Up environment is examined across various complexity levels to demonstrate how agent performance varies with complexity from as mentioned in [DALM⁺21]. Following this, the results for the Gym-DC environment are provided in Section 5.9. A discussion of these results is then presented in Section 5.10.

5.8.1 Scaling the Challenge: Agent Performance from easy to hard

The authors of [DALM⁺21] provided distinct settings for multiple complexity levels for several RL problems, including the Cart-Pole Swing-Up environment. A detailed description of the Swing-Up environment can be found in Appendix 8.5. The specific configurations for the easy, medium and hard levels of this problem are detailed in table 2.3.

5.8.2 Reinforcement Learning Agent Configuration

The RL agent was configured and trained based on the provided script and Specifications which are listed in table 5.34. The agent utilizes the Deep Deterministic Policy Gradient (DDPG) algorithm from the Stable-Baselines3 library [hb21].

5.8.3 Results

Two RL agents were trained using the algorithm described in table 5.34: one in an easy setting environment and another in a hard setting environment.

The performance of each agent was then evaluated over 1000 episodes within its respective training environment. The average rewards achieved in both scenarios are presented in table 5.35.

Table 5.34: RL Agent Specifications

Component / Hyperparameter	Value / Description
Algorithm & Framework	
RL Framework	Stable-Baselines3
Algorithm	DDPG
Policy Network	MlpPolicy
Total Training Timesteps	20,000,000
Action Noise	Gaussian ('NormalActionNoise')
Noise Standard Deviation (σ)	0.1
Software Versions (Assumed based on typical environment)	
Python	3.8.18
PyTorch	1.12.1+cu113
Stable-Baselines3	1.8.0
OpenAI Gym	0.21.0
NumPy	1.24.4

Table 5.35: Average 1000-Episode Reward Across Different Complexity Levels.

Level of Complexity	Average 1000-Episode Reward
Easy	189.643
Hard	82.703

The results indicate a significant decrease in performance for the hard setting, with its average reward of 83 being 56.3% lower than the reward from the easy setting. In the next sections, the performance of the RL agents in Gym-DC environment will be evaluated under the settings detailed in Chapter 5. This analysis will demonstrate the performance differences across various levels of complexity, similar to the approach used for the Cart-Pole Swing-Up environment in table 5.35.

5.9 Performance Analysis for Simulation Settings in different scenarios

This section presents a performance analysis for Simulation Settings 1 to 5 under three distinct scenarios: "On-time Delivery", "Random Delay", and "Unreliable Service Provider". The RL agent was trained using the algorithm detailed in table 5.36, which also lists its hyperparameters. The RL agent was implemented using the Stable-Baselines3 v1.8.0 library, which is built upon PyTorch v1.12.1 [hb21].

Table 5.36: RL Agent and Environment Specifications

Component / Hyperparameter	Value / Description
Algorithm & Framework	
RL Framework	Stable-Baselines3
Algorithm	DDPG
Policy Network	MlpPolicy
Total Timesteps	200,010
Action Noise	NormalActionNoise
Noise Standard Deviation (σ)	0.1
Software Versions	
Python	3.8.18
PyTorch	1.12.1+cu113
Stable-Baselines3	1.8.0
OpenAI Gym	0.21.0
NumPy	1.24.4

The experiments were conducted on a **Dell PowerEdge C4130** server. A detailed summary of the server's hardware specifications is presented in Table 5.37.

Table 5.37: Server Hardware Specifications

Component	Specification
Server Model	Dell PowerEdge C4130
CPU	
Model	2 x Intel(R) Xeon(R) E5-2630 v4 @ 2.20GHz
Architecture	x86_64
Cores/Threads	20 Cores / 40 Threads (Total)
L1 Cache	640 KiB per CPU
L2 Cache	2.5 MiB per CPU
L3 Cache	25 MiB per CPU
GPU	
Model	8 x NVIDIA Tesla K80
GPU Memory (VRAM)	12 GB per GPU (Total 96 GB)
NVIDIA Driver	470.256.02
CUDA Version	11.4
Memory & Storage	
System Memory (RAM)	128 GiB DDR4 @ 2133 MHz
Storage	2 x 800 GB Intel SSD

To facilitate a clearer comparison of results across different heuristics and RL agent, we evaluated the performance using 5000 different seeds for the random generator in the simulator. To improve readability and comparability, we applied a moving average technique with a window size of 20. This method smooths the performance data by averaging values over the specified window size, producing a clearer visual comparison of the results. For each setting and scenario, we include two tables and one plot. The first table reports the pairwise relative performance of different baselines across 5000 simulation runs for a given setting and scenario. Each cell $[i][j]$ represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j . The second table presents the number of seeds for which the row baseline strictly outperforms the column baseline. This table is shown specifically for Setting 1 under the "On-time Delivery" scenario. Cells represent the number of seeds with higher values for the row baseline.. Following the two tables, we provide a plot that visually compares the performance of the baselines. In Appendix 8.2, another table for each setting displays the Selective Performance Comparison Matrix, where each cell $[i][j]$ represents the ratio of the mean performance value in cases where baseline i outperforms baseline j across 5000 simulation runs. This approach highlights instances where strategy i demonstrates superior performance over strategy j at corresponding indices. The caption of each table specifies the simulation setting and scenario.

While all tables and plots provide information to compare the baselines with each other, the main discussion and analysis of these results are presented in Section 5.10.

Performance Analysis for Simulation Setting1 in On-time delivery

Table 5.38: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting1, On-time delivery. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	100.00	92.43	78.83	67.09	62.71	61.77	61.77
Conservative		100.00	85.28	72.59	67.85	66.82	66.82
Cautious			100.00	85.12	79.56	78.36	78.36
Stable				100.00	93.47	92.06	92.06
Optimistic					100.00	98.49	98.49
Greedy						100.00	100.00
RL							100.00

Table 5.39: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting1 in On-time delivery. Cells represent the number of seeds with higher values for the row baseline.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	0	35	9	0	0	0	0
Conservative	4620	0	93	2	0	0	0
Cautious	4690	4593	0	42	4	0	0
Stable	4740	4739	4647	0	38	0	0
Optimistic	4806	4806	4749	3229	0	0	0
Greedy	4827	4827	4775	3292	691	0	0
RL	4827	4827	4775	3292	691	0	0

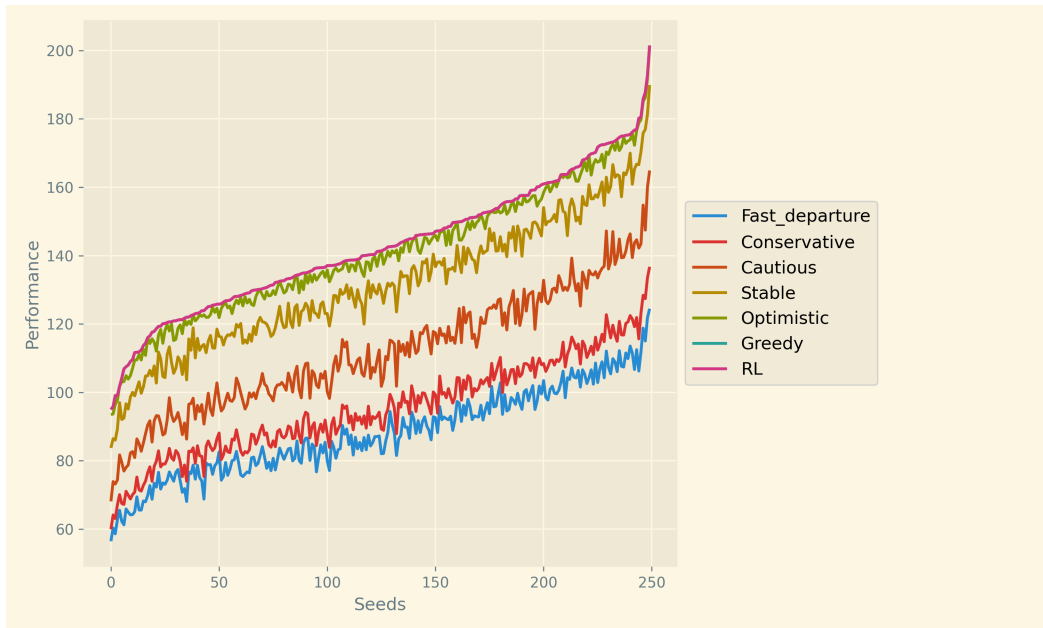


Figure 5.20: Plotting performance of different baselines for Setting1 in On-time delivery window size = 20

Performance Analysis for Simulation Setting1 in Random delay

Table 5.40: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting1, Random delay. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	100.00	94.81	84.76	75.29	71.50	70.65	70.65
Conservative		100.00	89.40	79.41	75.42	74.52	74.52
Cautious			100.00	88.83	84.36	83.36	83.36
Stable				100.00	94.97	93.84	93.84
Optimistic					100.00	98.81	98.81
Greedy						100.00	100.00
RL							100.00

Table 5.41: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting1 in Random delay. Cells represent the number of seeds with higher values for the row baseline.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	0	1051	322	56	14	11	11
Conservative	3811	0	580	102	26	16	16
Cautious	4548	4291	0	263	60	42	42
Stable	4834	4787	4612	0	379	303	303
Optimistic	4906	4894	4844	3304	0	126	126
Greedy	4930	4925	4882	3401	807	0	0
RL	4930	4925	4882	3401	807	0	0

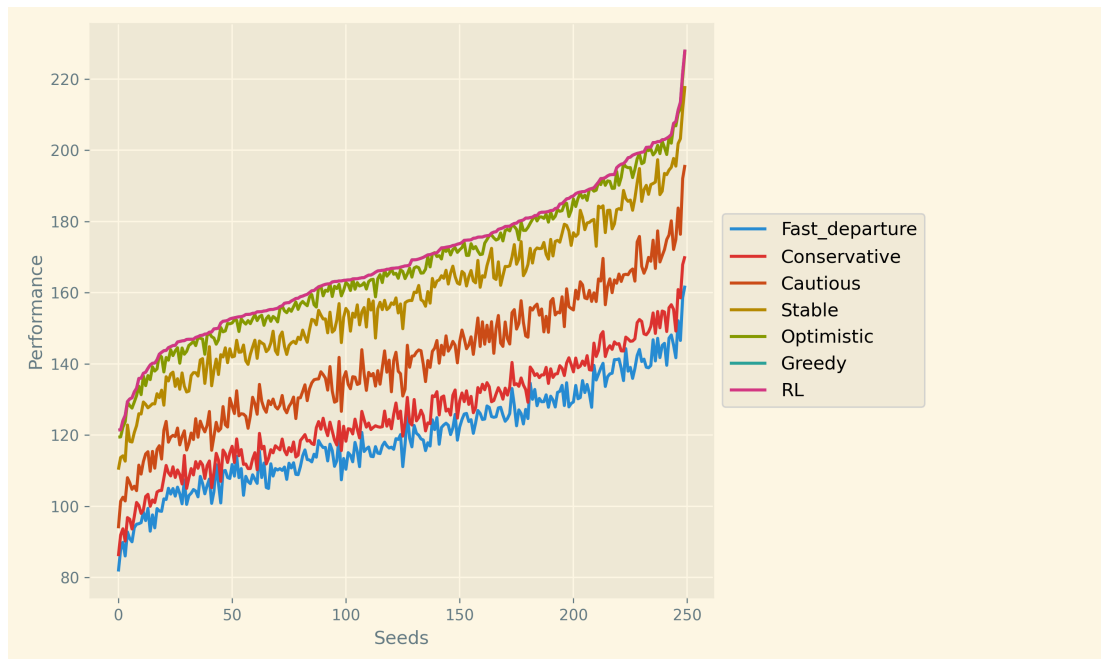


Figure 5.21: Plotting performance of different baselines for Setting1 in Random delay window size = 20

Performance Analysis for Simulation Setting1 in Unreliable service provider

Table 5.42: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting1, Unreliable service provider. Each cell $[i][j]$ represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j .

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	100.00	93.85	81.94	71.12	66.97	66.13	66.13
Conservative		100.00	87.31	75.78	71.36	70.47	70.47
Cautious			100.00	86.80	81.73	80.71	80.71
Stable				100.00	94.16	92.99	92.99
Optimistic					100.00	98.76	98.76
Greedy						100.00	100.00
RL							100.00

Table 5.43: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting1 in Unreliable service provider. Cells represent the number of seeds with higher values for the row baseline.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	0	424	301	177	141	150	150
Conservative	4281	0	354	157	135	137	137
Cautious	4446	4377	0	190	129	128	128
Stable	4597	4617	4527	0	172	135	135
Optimistic	4689	4695	4645	3084	0	36	36
Greedy	4697	4710	4664	3142	646	0	0
RL	4697	4710	4664	3142	646	0	0

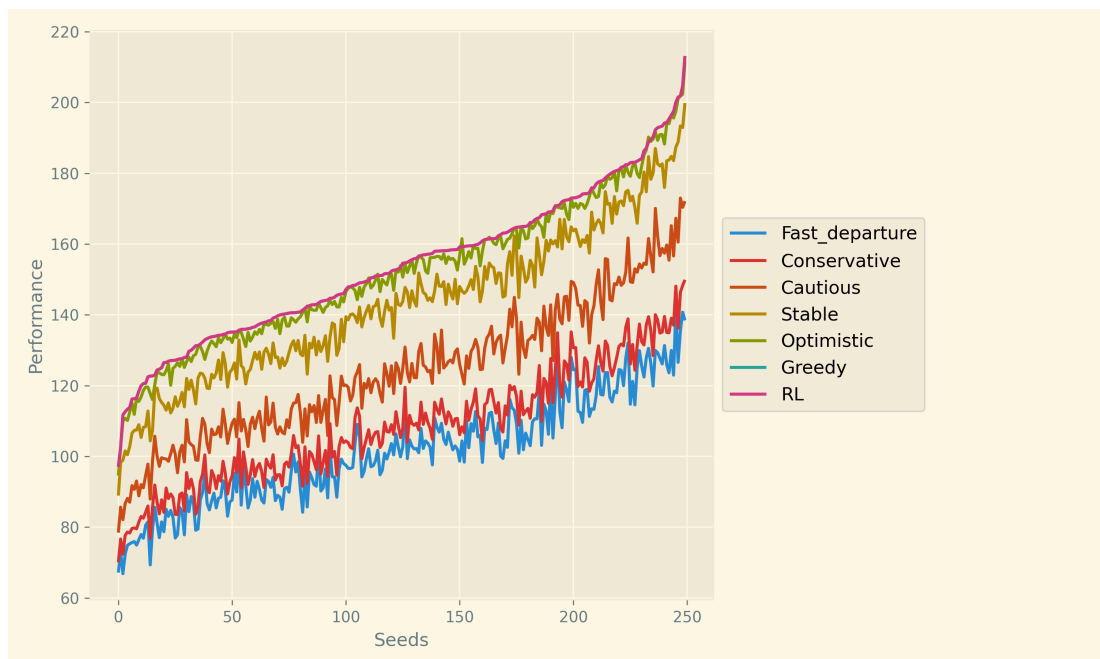


Figure 5.22: Plotting performance of different baselines for Setting1 in Unreliable service provider window size = 20

Performance Analysis for Simulation Setting2 in On-time delivery

Table 5.44: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting2, On-time delivery. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	100.00	97.56	99.61	99.83	99.83	99.83	97.04
Conservative		100.00	102.10	102.33	102.33	102.33	99.47
Cautious			100.00	100.23	100.23	100.23	97.42
Stable				100.00	100.00	100.00	97.20
Optimistic					100.00	100.00	97.20
Greedy						100.00	97.20
RL							100.00

Table 5.45: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting2 in On-time delivery. Cells represent the number of seeds with higher values for the row baseline.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	0	100	2116	2297	2297	2297	50
Conservative	4900	0	4681	4754	4754	4754	1175
Cautious	2884	296	0	1107	1107	1107	130
Stable	2703	223	176	0	0	0	54
Optimistic	2703	223	176	0	0	0	54
Greedy	2703	223	176	0	0	0	54
RL	4947	3825	4870	4946	4946	4946	0

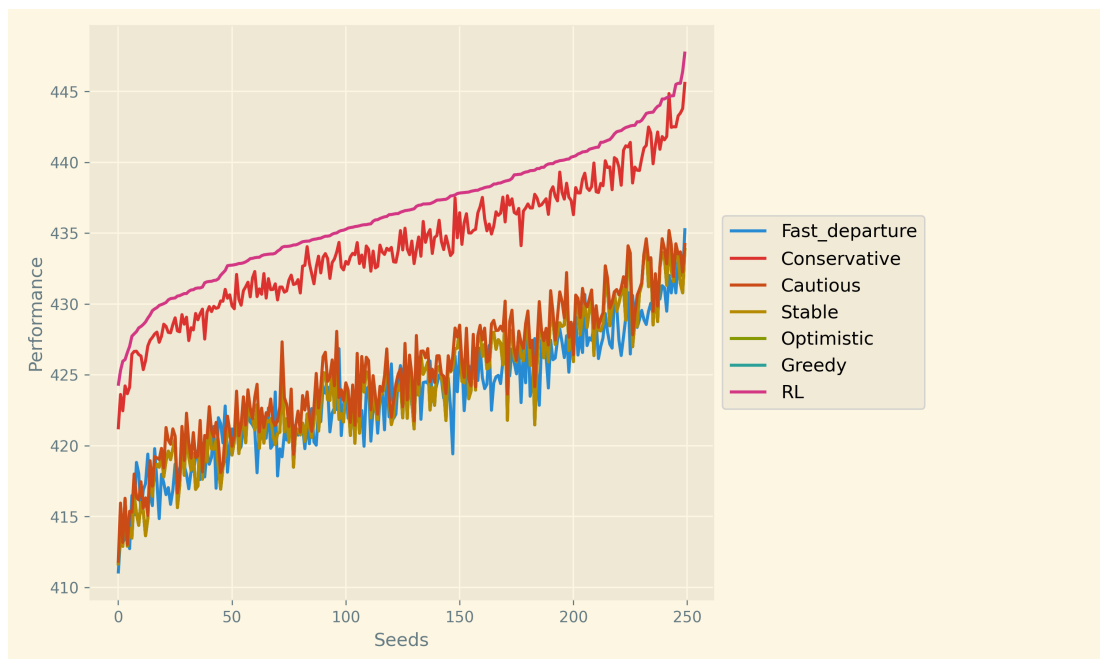


Figure 5.23: Plotting performance of different baselines for Setting2 in On-time delivery window size = 20

Performance Analysis for Simulation Setting2 in Random delay

Table 5.46: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting2, Random delay. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	100.00	97.66	99.74	99.97	99.97	99.97	96.69
Conservative		100.00	102.13	102.37	102.37	102.37	99.00
Cautious			100.00	100.23	100.23	100.23	96.94
Stable				100.00	100.00	100.00	96.71
Optimistic					100.00	100.00	96.71
Greedy						100.00	96.71
RL							100.00

Table 5.47: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting2 in Random delay. Cells represent the number of seeds with higher values for the row baseline.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	0	342	2227	2414	2414	2414	78
Conservative	4658	0	4428	4532	4532	4532	1094
Cautious	2773	550	0	952	952	952	228
Stable	2586	446	331	0	0	0	185
Optimistic	2586	446	331	0	0	0	185
Greedy	2586	446	331	0	0	0	185
RL	4919	3906	4772	4815	4815	4815	0

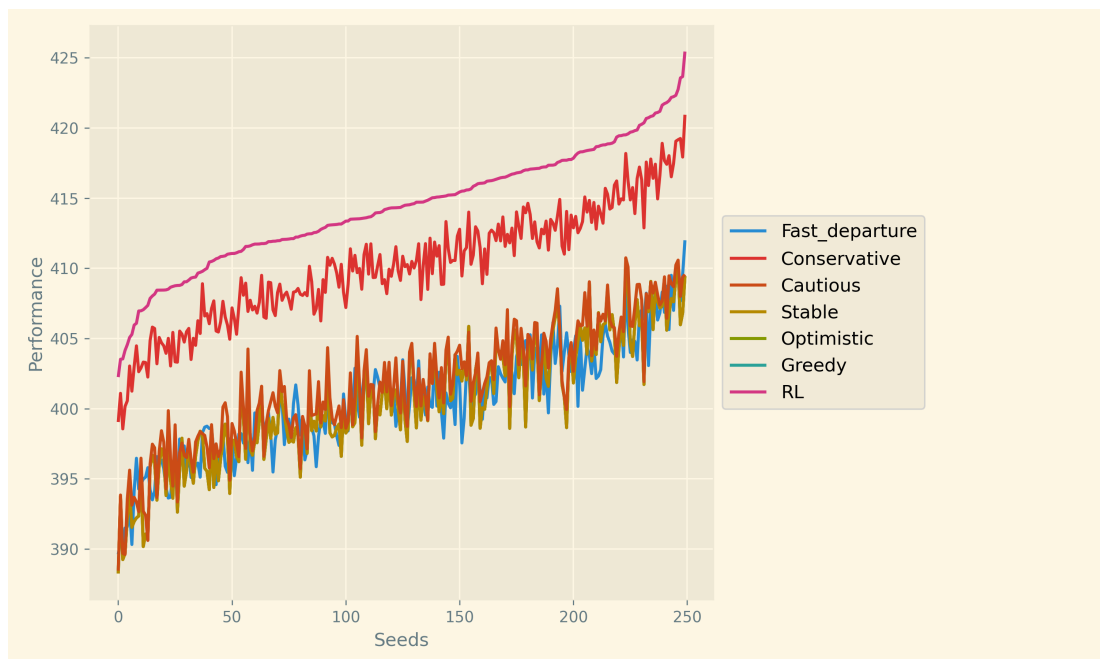


Figure 5.24: Plotting performance of different baselines for Setting2 in Random delay window size = 20

Performance Analysis for Simulation Setting2 in Unreliable service provider

Table 5.48: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting2, Unreliable service provider. Each cell $[i][j]$ represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j .

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	100.00	97.57	99.61	99.83	99.83	99.83	96.88
Conservative		100.00	102.10	102.32	102.32	102.32	99.30
Cautious			100.00	100.22	100.22	100.22	97.26
Stable				100.00	100.00	100.00	97.04
Optimistic					100.00	100.00	97.04
Greedy						100.00	97.04
RL							100.00

Table 5.49: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting2 in Unreliable service provider. Cells represent the number of seeds with higher values for the row baseline.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	0	408	2167	2318	2318	2318	497
Conservative	4592	0	4480	4558	4558	4558	977
Cautious	2833	496	0	1071	1071	1071	353
Stable	2681	418	220	0	0	0	325
Optimistic	2681	418	220	0	0	0	325
Greedy	2681	418	220	0	0	0	325
RL	4499	4023	4647	4675	4675	4675	0

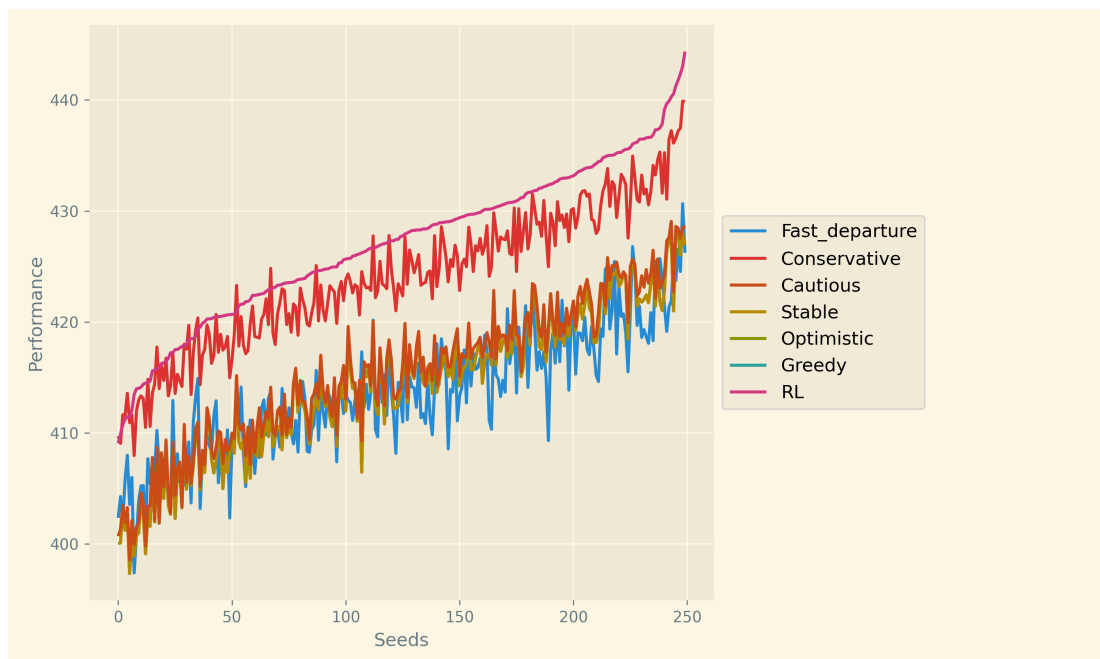


Figure 5.25: Plotting performance of different baselines for Setting2 in Unreliable service provider window size = 20

Performance Analysis for Simulation Setting3 in On-time delivery

Table 5.50: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting3, On-time delivery. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	100.00	107.66	107.66	107.66	107.66	107.66	98.35
Conservative		100.00	100.00	100.00	100.00	100.00	91.35
Cautious			100.00	100.00	100.00	100.00	91.35
Stable				100.00	100.00	100.00	91.35
Optimistic					100.00	100.00	91.35
Greedy						100.00	91.35
RL							100.00

Table 5.51: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting3 in On-time delivery. Cells represent the number of seeds with higher values for the row baseline.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	0	4872	4872	4872	4872	4872	386
Conservative	128	0	5	5	5	5	9
Cautious	128	2	0	0	0	0	9
Stable	128	2	0	0	0	0	9
Optimistic	128	2	0	0	0	0	9
Greedy	128	2	0	0	0	0	9
RL	4609	4991	4991	4991	4991	4991	0

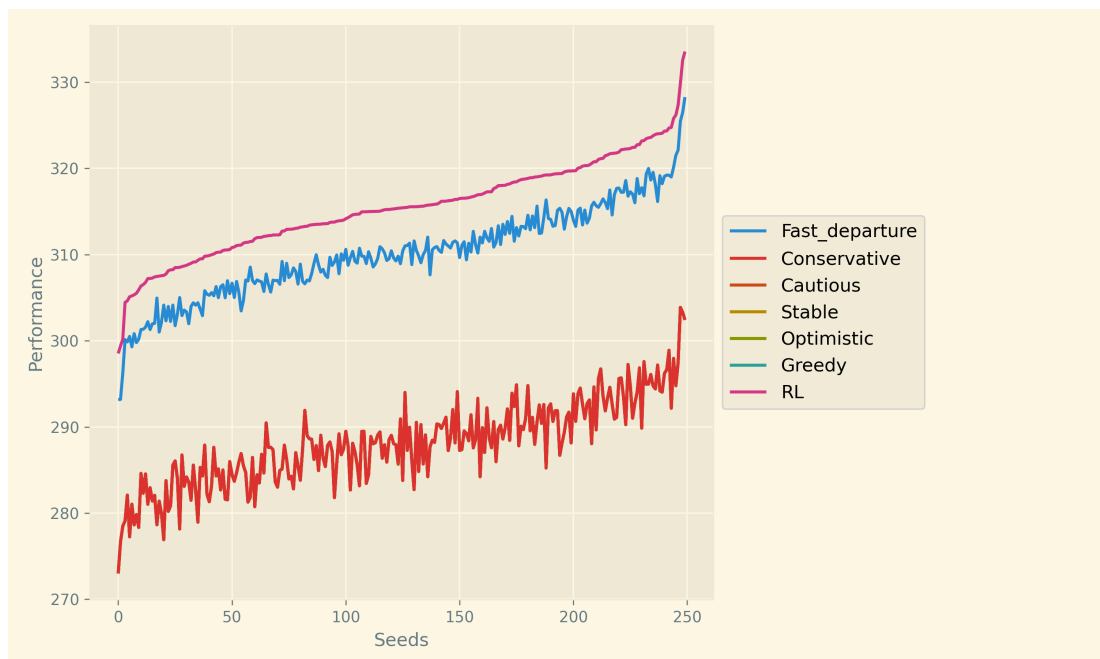


Figure 5.26: Plotting performance of different baselines for Setting3 in On-time delivery window size = 20

Performance Analysis for Simulation Setting3 in Random delay

Table 5.52: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting3, Random delay. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	100.00	107.96	107.96	107.96	107.96	107.96	98.41
Conservative		100.00	100.00	100.00	100.00	100.00	91.15
Cautious			100.00	100.00	100.00	100.00	91.15
Stable				100.00	100.00	100.00	91.15
Optimistic					100.00	100.00	91.15
Greedy						100.00	91.15
RL							100.00

Table 5.53: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting3 in Random delay. Cells represent the number of seeds with higher values for the row baseline.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	0	4871	4871	4871	4871	4871	1012
Conservative	129	0	5	5	5	5	10
Cautious	129	2	0	0	0	0	10
Stable	129	2	0	0	0	0	10
Optimistic	129	2	0	0	0	0	10
Greedy	129	2	0	0	0	0	10
RL	3983	4990	4990	4990	4990	4990	0

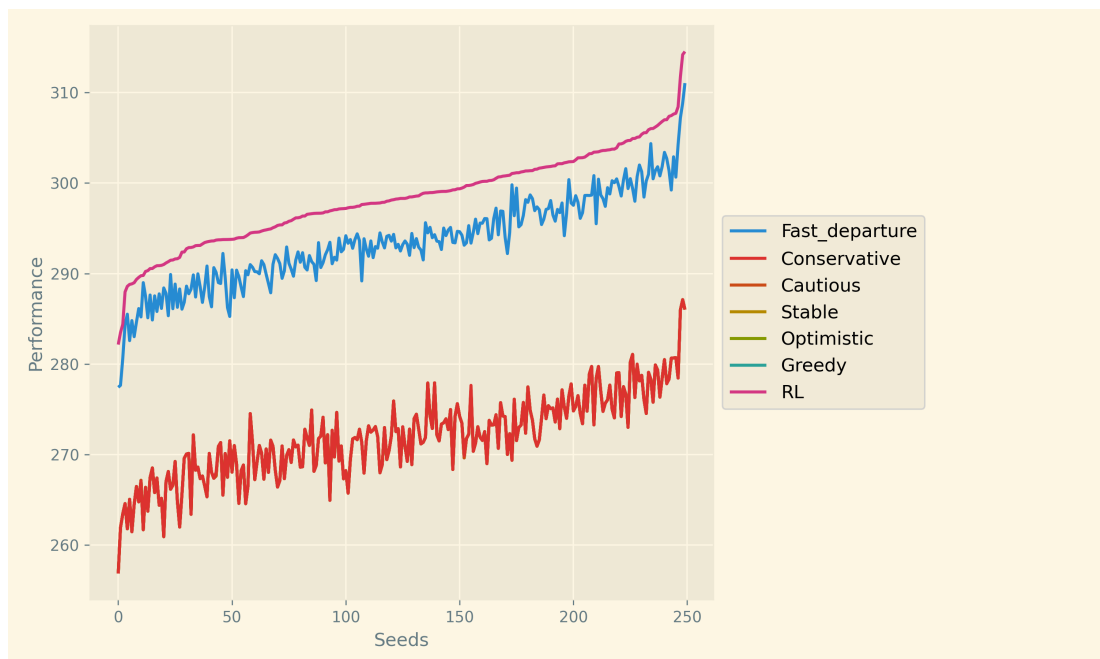


Figure 5.27: Plotting performance of different baselines for Setting3 in Random delay window size = 20

Performance Analysis for Simulation Setting3 in Unreliable service provider

Table 5.54: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting3, Unreliable service provider. Each cell $[i][j]$ represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j .

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	100.00	107.69	107.69	107.69	107.69	107.69	98.95
Conservative		100.00	100.00	100.00	100.00	100.00	91.88
Cautious			100.00	100.00	100.00	100.00	91.88
Stable				100.00	100.00	100.00	91.88
Optimistic					100.00	100.00	91.88
Greedy						100.00	91.88
RL							100.00

Table 5.55: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting3 in Unreliable service provider. Cells represent the number of seeds with higher values for the row baseline.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	0	4659	4659	4659	4659	4659	734
Conservative	340	0	5	5	5	5	127
Cautious	340	2	0	0	0	0	127
Stable	340	2	0	0	0	0	127
Optimistic	340	2	0	0	0	0	127
Greedy	340	2	0	0	0	0	127
RL	4263	4873	4873	4873	4873	4873	0

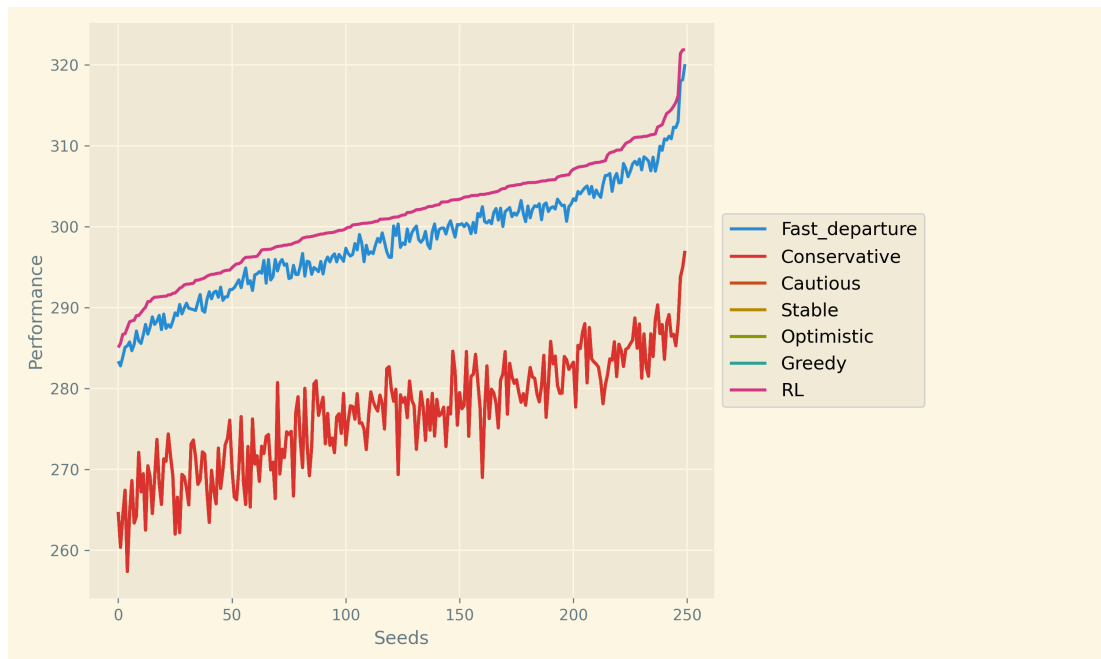


Figure 5.28: Plotting performance of different baselines for Setting3 in Unreliable service provider window size = 20

Performance Analysis for Simulation Setting4 in On-time delivery

Table 5.56: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting4, On-time delivery. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	100.00	100.41	101.95	103.08	101.72	98.71	95.97
Conservative		100.00	101.53	102.66	101.31	98.31	95.58
Cautious			100.00	101.11	99.78	96.82	94.14
Stable				100.00	98.68	95.76	93.11
Optimistic					100.00	97.04	94.35
Greedy						100.00	97.23
RL							100.00

Table 5.57: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting4 in On-time delivery. Cells represent the number of seeds with higher values for the row baseline.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	0	2940	4238	4258	3296	1849	770
Conservative	1439	0	4120	4190	3160	1701	641
Cautious	736	813	0	3441	2322	1106	322
Stable	742	810	1558	0	1469	656	173
Optimistic	1704	1840	2678	3530	0	724	160
Greedy	3151	3299	3894	4344	4215	0	477
RL	4230	4359	4678	4827	4840	4523	0

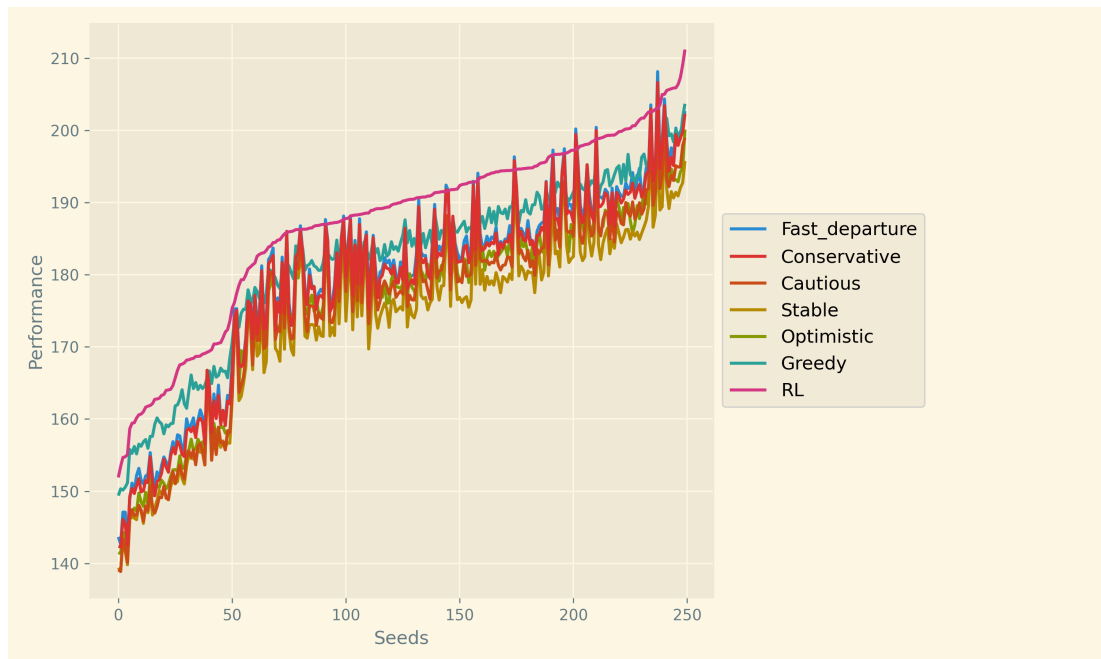


Figure 5.29: Plotting performance of different baselines for Setting4 in On-time delivery window size = 20

Performance Analysis for Simulation Setting4 in Random delay

Table 5.58: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting4, Random delay. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	100.00	100.47	102.24	103.81	103.13	100.88	97.63
Conservative		100.00	101.76	103.33	102.65	100.41	97.18
Cautious			100.00	101.54	100.87	98.67	95.49
Stable				100.00	99.35	97.18	94.05
Optimistic					100.00	97.82	94.67
Greedy						100.00	96.78
RL							100.00

Table 5.59: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting4 in Random delay. Cells represent the number of seeds with higher values for the row baseline.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	0	2739	4187	4473	3909	2756	1093
Conservative	1670	0	4001	4358	3780	2549	980
Cautious	787	936	0	3636	2958	1819	564
Stable	527	642	1363	0	1977	1087	361
Optimistic	1091	1220	2042	3022	0	1140	427
Greedy	2244	2451	3181	3913	3801	0	940
RL	3907	4020	4436	4639	4573	4060	0

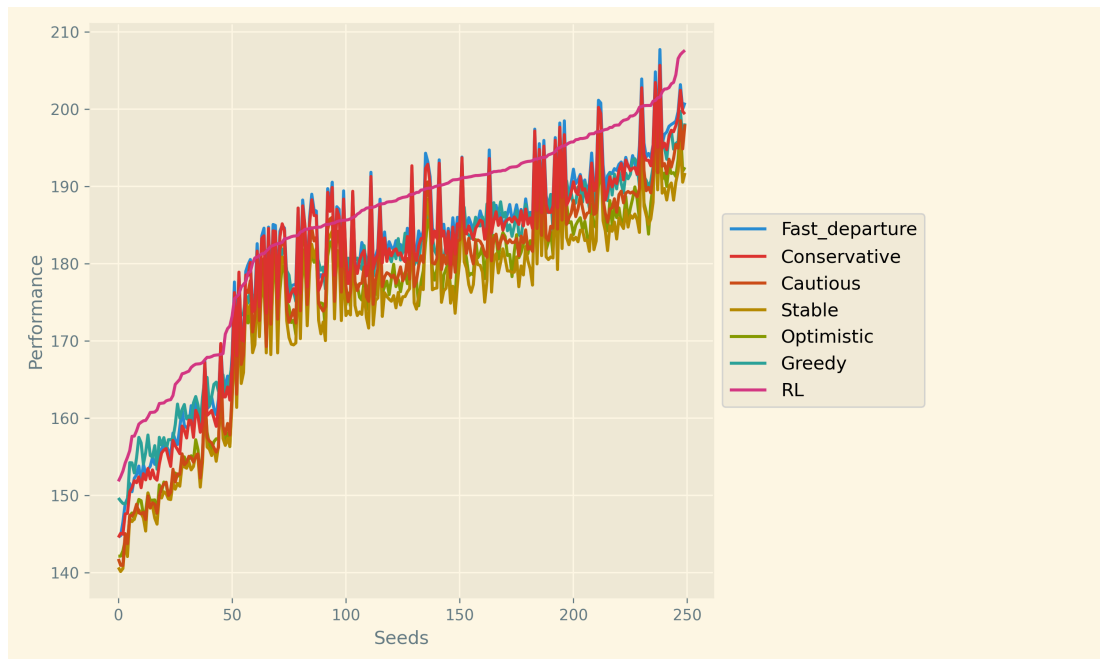


Figure 5.30: Plotting performance of different baselines for Setting4 in Random delay window size = 20

Performance Analysis for Simulation Setting4 in Unreliable service provider

Table 5.60: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting4, Unreliable service provider. Each cell $[i][j]$ represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j .

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	100.00	100.46	102.12	103.49	102.48	99.93	96.81
Conservative		100.00	101.66	103.01	102.01	99.47	96.37
Cautious			100.00	101.34	100.35	97.85	94.80
Stable				100.00	99.03	96.56	93.55
Optimistic					100.00	97.51	94.46
Greedy						100.00	96.88
RL							100.00

Table 5.61: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting4 in Unreliable service provider. Cells represent the number of seeds with higher values for the row baseline.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	0	2978	4320	4364	3608	2330	1156
Conservative	1394	0	4176	4284	3456	2157	1066
Cautious	657	756	0	3599	2648	1543	746
Stable	636	716	1400	0	1721	960	482
Optimistic	1392	1544	2351	3278	0	976	476
Greedy	2670	2843	3457	4040	3951	0	820
RL	3844	3934	4254	4518	4524	4175	0

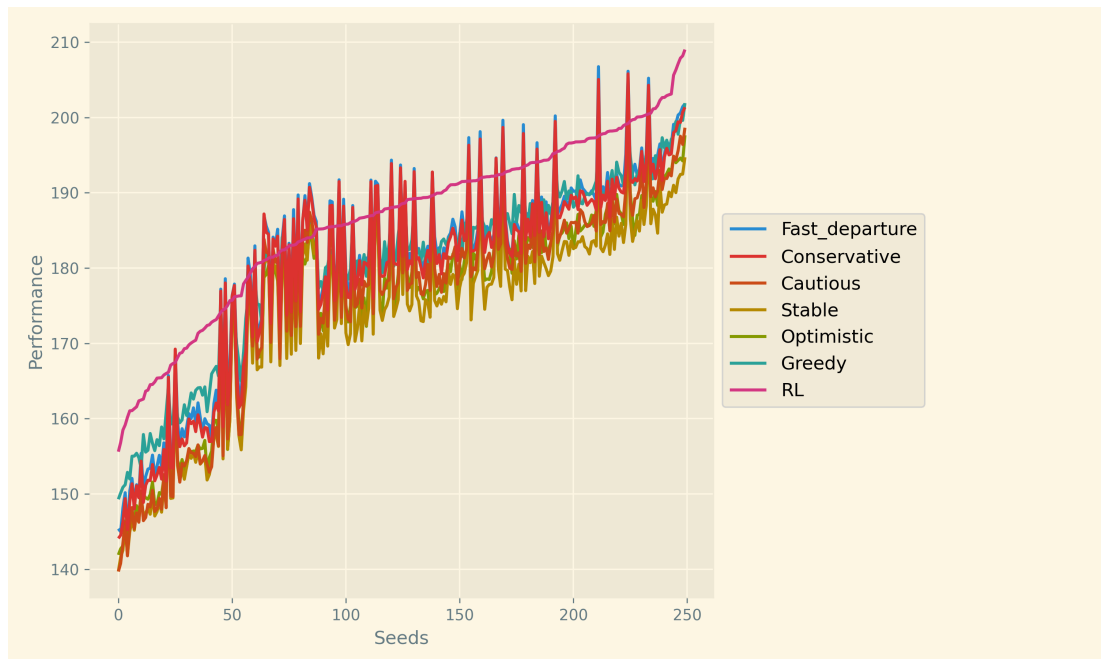


Figure 5.31: Plotting performance of different baselines for Setting4 in Unreliable service provider window size = 20

Performance Analysis for Simulation Setting5 in On-time delivery

Table 5.62: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting5, On-time delivery. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	100.00	98.03	94.01	92.31	92.06	92.69	90.86
Conservative		100.00	95.90	94.17	93.91	94.56	92.69
Cautious			100.00	98.19	97.93	98.60	96.65
Stable				100.00	99.73	100.42	98.43
Optimistic					100.00	100.69	98.70
Greedy						100.00	98.02
RL							100.00

Table 5.63: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting5 in On-time delivery. Cells represent the number of seeds with higher values for the row baseline.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	0	257	93	152	317	573	271
Conservative	4728	0	242	335	546	848	447
Cautious	4907	4758	0	1090	1352	1712	867
Stable	4848	4665	3909	0	2160	2529	1239
Optimistic	4683	4454	3646	2793	0	2568	1257
Greedy	4427	4152	3288	2432	1752	0	844
RL	4729	4553	4133	3760	3742	4155	0

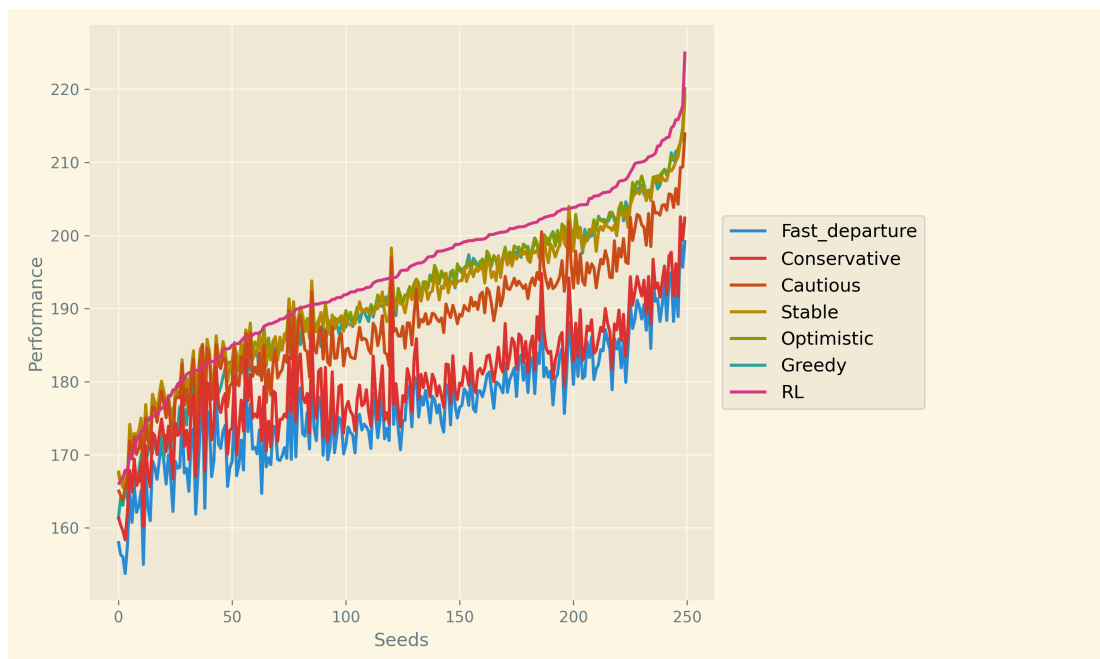


Figure 5.32: Plotting performance of different baselines for Setting5 in On-time delivery window size = 20

Performance Analysis for Simulation Setting5 in Random delay

Table 5.64: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting5, Random delay. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	100.00	98.25	94.85	93.67	93.78	94.64	92.69
Conservative		100.00	96.54	95.34	95.46	96.33	94.34
Cautious			100.00	98.75	98.87	99.77	97.72
Stable				100.00	100.12	101.03	98.95
Optimistic					100.00	100.91	98.83
Greedy						100.00	97.94
RL							100.00

Table 5.65: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting5 in Random delay. Cells represent the number of seeds with higher values for the row baseline.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	0	712	219	324	553	908	370
Conservative	4278	0	505	584	870	1227	611
Cautious	4781	4493	0	1589	1874	2225	1299
Stable	4676	4415	3409	0	2571	2950	1761
Optimistic	4446	4130	3125	2393	0	2727	1660
Greedy	4092	3773	2773	2017	1599	0	1198
RL	4629	4388	3698	3236	3337	3800	0

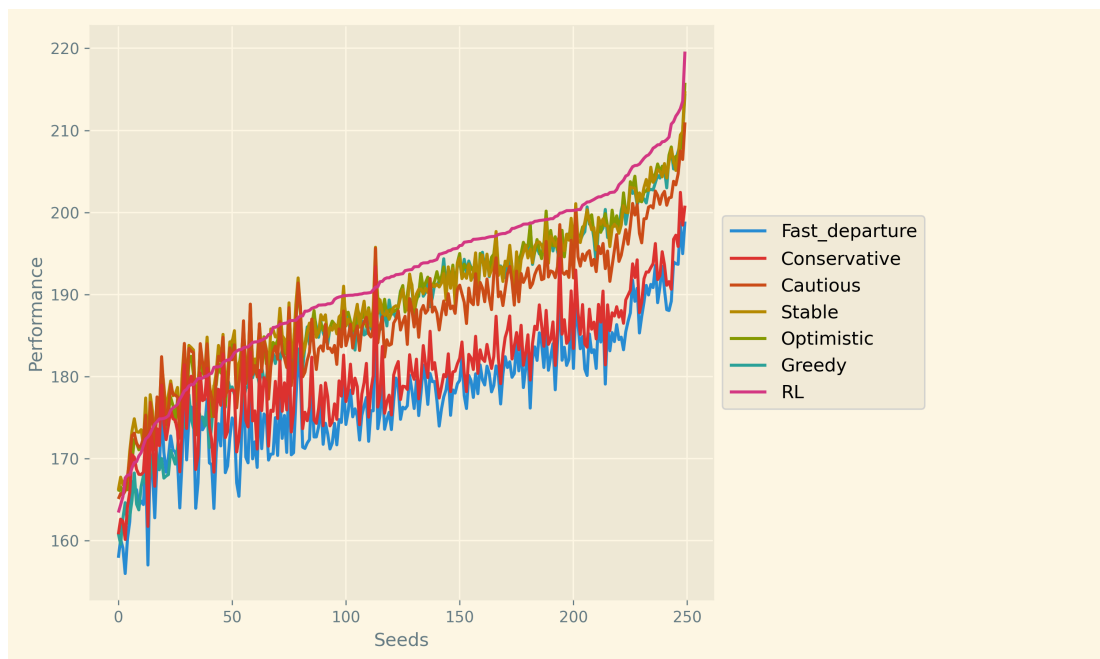


Figure 5.33: Plotting performance of different baselines for Setting5 in Random delay window size = 20

Performance Analysis for Simulation Setting5 in Unreliable service provider

Table 5.66: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting5, Unreliable service provider. Each cell $[i][j]$ represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j .

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	100.00	98.16	94.58	93.14	93.12	93.91	91.82
Conservative		100.00	96.35	94.88	94.87	95.67	93.55
Cautious			100.00	98.47	98.46	99.29	97.09
Stable				100.00	99.98	100.83	98.59
Optimistic					100.00	100.85	98.61
Greedy						100.00	97.78
RL							100.00

Table 5.67: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting5 in Unreliable service provider. Cells represent the number of seeds with higher values for the row baseline.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast_departure	0	382	215	351	574	869	414
Conservative	4603	0	404	577	841	1175	616
Cautious	4785	4595	0	1358	1665	2033	1071
Stable	4649	4423	3639	0	2418	2771	1453
Optimistic	4426	4159	3333	2540	0	2739	1384
Greedy	4131	3824	2967	2187	1573	0	948
RL	4584	4383	3927	3545	3614	4052	0

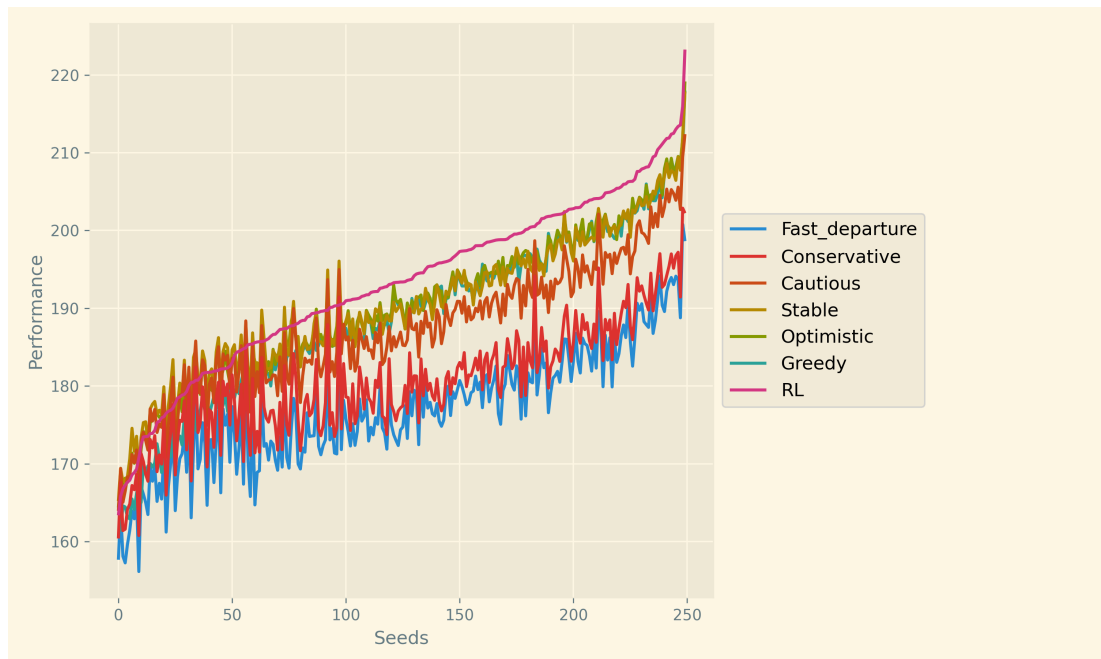


Figure 5.34: Plotting performance of different baselines for Setting5 in Unreliable service provider window size = 20

5.10 Discussion

The results demonstrate a clear performance advantage for the RL agent, particularly in settings 1,2 and 3. In **Setting 1**, the agent's average performance was indistinguishable from the top-performing **Greedy** baseline, with both strategies significantly surpassing all alternatives. In **Settings 2 and 3**, the RL agent emerged as distinctly superior to all baselines. This superiority is evident not only in its higher average performance but also in its consistency in achieving optimal outcomes. As detailed in Table 5.69, the RL agent attained the maximum score in the vast majority of simulation runs for these settings, frequently in over 80-90% of instances. However, this pattern of dominance moderates in the **Settings 4 and 5**. While the RL agent continues to yield the highest *average* performance, its frequency of securing the absolute best score in a given run decreases markedly. For example, in **Setting 4**, this frequency diminishes across all scenarios, notably dropping to approximately **62%** in the random delay scenario. The trend is further pronounced in **Setting 5**, where the agent's ability to achieve the top score is significantly reduced, falling to just **49%** under the same random delay conditions.

The experimental results indicate that settings 1, 2, and 3 can be classified as having a low level of complexity, while settings 4 and 5 represent a high level of complexity. In addition to classifying settings 1-3 as low-complexity and 4-5 as high-complexity, the experiments confirmed that the conditions for the expected value of the reward function, outlined in Section 3.2.5, can be verified during the RL agent's training in settings 4 and 5. The packet rate is set to a fluctuation mode in Settings 4 and 5. This means the system's final performance value will vary based on the different rates of incoming packets, which is the key insight into the RL agents' performance under these conditions. Table 5.69 indicates that setting 5 under the random delay scenario represents the most complex configuration. Nevertheless, an alternative classification based on average numerical values, distinguishing between "easy" and "hard" settings, is presented in Table 5.68.

Table 5.68: Classification of Environment Settings by Complexity Level.

Level of Complexity	Environment setting
Easy	Settings 1,2 and 3
Hard	Settings 4 and 5

Table 5.69: Performance Comparison of Baselines and RL Agent. The table shows the frequency, out of 5000 runs, that each strategy attained the maximum performance score for a given experimental setting.

Scenario and Setting	Fast_departure	Greedy	Optimistic	Stable	Cautious	Conservative	RL
On-time delivery							
Setting 1	173	5000	4309	1708	225	173	5000
Setting 2	15	15	15	15	47	1151	3786
Setting 3	391	8	8	8	8	8	4606
Setting 4	597	369	46	18	33	227	3803
Setting 5	9	471	536	627	426	82	2947
Random delay							
Setting 1	66	4529	3855	1564	110	70	4529
Setting 2	44	85	85	85	123	1026	3802
Setting 3	1016	9	9	9	9	9	3980
Setting 4	681	699	101	46	94	425	3104
Setting 5	20	505	659	814	556	137	2441
Unreliable service provider							
Setting 1	229	4738	4137	1759	244	187	4738
Setting 2	320	30	30	30	49	671	3960
Setting 3	730	122	122	122	122	122	4151
Setting 4	755	503	75	33	84	368	3344
Setting 5	48	454	565	705	476	146	2726

5.11 Conclusion

This chapter introduced GymDC, a reinforcement learning environment built to simulate the operations of a distribution center (DC) within the broader framework of the Physical Internet. The system models packets with attributes such as weight, delivery budgets, and penalty costs, while also representing service providers with differing capacities, costs, and schedules. These features make the environment suitable for developing and testing reinforcement learning algorithms in situations where decisions must be made sequentially and often without full information. Several baseline policies were implemented to provide comparisons for future reinforcement learning agents. These include a profit-driven greedy strategy, a deadline-sensitive fast departure policy, and a hybrid approach that switches between the two based on current storage capacity. Their performance was tested across a variety of settings and scenarios, ranging from ideal delivery conditions to environments with random delays or unreliable service providers. In summary, GymDC provides a flexible platform for studying reinforcement learning in supply chain environments. It offers a foundation for experimenting with policy design, performance evaluation, and learning under uncertainty. By simulating the trade-offs between profitability and service reliability, GymDC encourages the development of smarter, more responsive agents for future logistics systems.

Furthermore, the chapter presented a performance analysis of a RL agent against baselines across five simulation settings. The investigation began by establishing a foundational understanding of how environmental complexity impacts agent performance, using the Cart-Pole Swing-Up problem as an illustrative example. This preliminary analysis confirmed that, as expected, performance degrades as the problem's difficulty increases. The core findings from the Gym-DC environment experiments revealed a clear pattern. In settings 1, 2, and 3, the RL agent demonstrated dominant performance, either matching the top-performing heuristic or substantially outperforming all baselines. However, as the complexity increased in Settings 4 and 5, the performance degrades and while the RL agent maintained its superiority in terms of average performance across all scenarios, it was unable to achieve the best score in all simulations. This was particularly evident in Setting 5 under the "Random delay" scenario—the most challenging configuration—where the agent's frequency of achieving the top score dropped significantly. This trend mirrors the initial observations from the Cart-Pole experiment, reinforcing the notion that increasing

complexity poses a substantial challenge to the RL agent's performance. Based on these experiments, we classified the settings and defined two levels of complexity for the Gym-DC environment. the RL agent's performance validates our classification of the Gym-DC environment. The agent achieved, on average, approximately 85% of the maximum performance in the easy class, whereas it only reached 55% in the hard class.

Chapter 6

Generative reward machine for GymDC

6.1 Introduction

As outlined in Chapter 5, the Gym-DC simulation environment incorporates five operational configurations designed to represent a diverse range of logistical complexities. Chapter 5 established the effectiveness of deep reinforcement learning (RL) agents within the first three settings, characterized by relatively stable and predictable conditions. However, Settings 4 and 5 introduce significant variability in packet arrival rates, compounded by the "Random Delay" and "Unreliable Service Provider" scenarios. These features result in noisy, non-stationary conditions known to undermine the performance of conventional RL agents. Given the complexity and stochastic nature of these latter settings, a detailed examination was reserved specifically for this chapter.

In this chapter, we directly engage with these challenging scenarios. Initially, we empirically validate the expected degradation in the performance of standard RL agents within Settings 4 and 5, underscoring the inherent limitations of traditional methods when confronted with pronounced environmental uncertainty. Subsequently, we propose and rigorously evaluate a novel Generative Reward Machine (GRM)-based agent. This chapter will elaborate on the methodological implementation of the GRM framework within the Gym-DC environment and provide a comparative analysis illustrating the enhanced robustness and superior performance of the GRM agent relative to conventional RL approaches under conditions of substantial stochasticity.

6.2 GRM Framework for the Gym-DC Environment

To address the challenges posed by the high environmental stochasticity in Settings 4 and 5, we propose a GRM-based agent. The core innovation of this approach lies in its ability to abstract the noisy, fluctuating patterns of daily packet arrivals into a stable, classified representation, which then informs the agent’s state and reward calculations.

6.2.1 Estimator for Packet Arrival Patterns

The foundational component of our GRM is an estimator designed to classify the temporal distribution of incoming packets. In environments where the packet arrival rate is set to ‘fluctuation’, the day-to-day volume of arrivals is highly variable, making it difficult for a standard RL agent to form a consistent policy. The estimator mitigates this by categorizing the observed arrival pattern into one of several predefined classes.

Estimator Training. The estimator is constructed through an offline analysis prior to the agent’s training phase. The procedure is as follows:

1. **Data Generation:** The Gym-DC simulator is executed 1,000 times using a setting with a fluctuating packet rate. For each execution, the sequence of total packets arriving per day over the entire simulation period is recorded.
2. **Clustering:** These 1,000 time-series vectors, representing distinct arrival patterns, are then grouped using a clustering algorithm to identify 20 distinct archetypal patterns, or classes. Each class represents a typical mode of fluctuation.
3. **Estimator Function:** The resulting cluster centroids serve as the basis for the estimator. During online operation, the estimator takes the sequence of packet arrivals observed up to the current day in an episode and classifies this partial pattern by finding the nearest cluster centroid. The output is a single integer, from 1 to 20, representing the estimated arrival pattern class for the current episode.

6.2.2 GRM State and Reward Formulation

With the estimator in place, the agent's state representation and reward function are augmented to leverage this new layer of abstraction.

State Space. The state space for the standard RL agent, as defined in Chapter 5, is a tuple consisting of the current 'storage capacity value' and 'rate'. The GRM agent employs an expanded state representation that incorporates the output of the packet pattern estimator. At each timestep t , the GRM state s'_t is defined as a tuple:

$$s'_t = (\text{storage_capacity_value}_t, \text{rate}_t, \text{estimated_pattern_class}_t) \quad (6.1)$$

Reward Generation. A significant challenge in noisy environments is reward variance, where a single good or bad outcome can disproportionately influence the policy update. To promote more stable learning, the GRM employs a reward averaging mechanism. Instead of using the direct performance value at the end of an episode as the reward, the GRM maintains a historical record of rewards received for each unique GRM state s' . When an episode terminates in state s' , the reward provided to the agent, $R(s')$, is the running average of all performance values previously recorded for that specific state:

$$R_t(s') = \frac{1}{N_{s'}} \sum_{i=1}^{N_{s'}} P_i(s') \quad (6.2)$$

where $N_{s'}$ is the number of times state s' has been visited, and $P_i(s')$ is the end-of-episode performance from the i -th visit. This reward-averaging approach smooths the learning signal, preventing the agent from overreacting to stochastic outliers and encouraging convergence towards a more generally effective strategy.

6.3 Performance Analysis for Simulation Settings in different scenarios

This section presents a performance analysis for Simulation Settings 4 and 5 under three distinct scenarios: "On-time Delivery", "Random Delay", and "Unreliable Service Provider". To facilitate a clearer comparison of results across different heuristics and RL agent and GRM agent, we evaluated the perfor-

mance using 5000 different seeds for the random generator in the simulator. To improve readability and comparability, we applied a moving average technique with a window size of 20. This method smooths the performance data by averaging values over the specified window size, producing a clearer visual comparison of the results. For each setting and scenario, we include two tables and one plot. The first table reports the pairwise relative performance of different baselines across 5000 simulation runs for a given setting and scenario. Each cell $[i][j]$ represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j . The second table presents the number of seeds for which the row baseline strictly outperforms the column baseline. This table is shown specifically for Setting 1 under the "On-time Delivery" scenario. Cells represent the number of seeds with higher values for the row baseline.. Following the two tables, we provide a plot that visually compares the performance of the baselines. In Appendix 8.3, another table for each setting displays the Selective Performance Comparison Matrix, where each cell $[i][j]$ represents the ratio of the mean performance value in cases where baseline i outperforms baseline j across 5000 simulation runs. This approach highlights instances where strategy i demonstrates superior performance over strategy j at corresponding indices. The caption of each table specifies the simulation setting and scenario.

While all tables and plots provide information to compare the baselines with each other, the main discussion and analysis of these results are presented in Section 6.4.

Performance Analysis for Simulation Setting 4 in On-time delivery

Table 6.1: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 4, On-time delivery.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL	Generative
Fast_departure	100.00	100.41	101.95	103.08	101.72	98.71	95.97	94.16
Conservative		100.00	101.53	102.66	101.31	98.31	95.58	93.78
Cautious			100.00	101.11	99.78	96.82	94.14	92.36
Stable				100.00	98.68	95.76	93.11	91.35
Optimistic					100.00	97.04	94.35	92.57
Greedy						100.00	97.23	95.40
RL							100.00	98.12
Generative								100.00

Table 6.2: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 4 in On-time delivery.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL	Generative
Fast_departure	0	2940	4238	4258	3296	1849	770	308
Conservative	1439	0	4120	4190	3160	1701	641	230
Cautious	736	813	0	3441	2322	1106	322	69
Stable	742	810	1558	0	1469	656	173	28
Optimistic	1704	1840	2678	3530	0	724	160	31
Greedy	3151	3299	3894	4344	4215	0	477	179
RL	4230	4359	4678	4827	4840	4523	0	879
Generative	4692	4770	4931	4972	4969	4821	4121	0

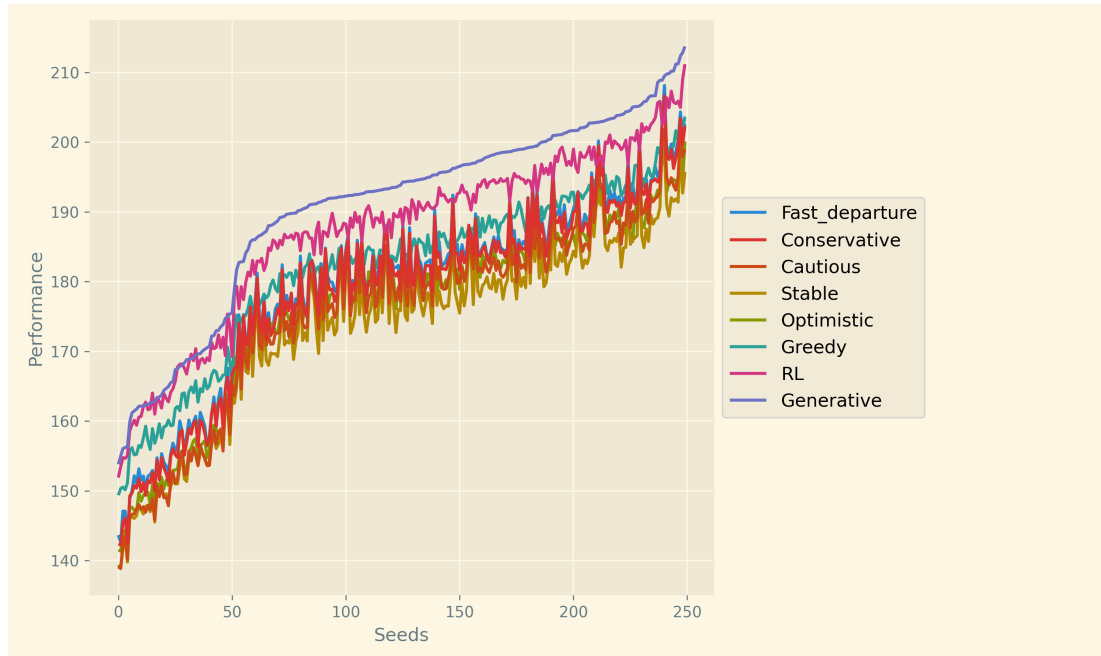


Figure 6.1: Plotting performance of different baselines for Setting 4 in On-time delivery window size = 20

Performance Analysis for Simulation Setting 4 in Random delay

Table 6.3: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 4, Random delay.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL	Generative
Fast_departure	100.00	100.47	102.24	103.81	103.13	100.88	97.63	95.85
Conservative		100.00	101.76	103.33	102.65	100.41	97.18	95.40
Cautious			100.00	101.54	100.87	98.67	95.49	93.74
Stable				100.00	99.35	97.18	94.05	92.32
Optimistic					100.00	97.82	94.67	92.93
Greedy						100.00	96.78	95.01
RL							100.00	98.17
Generative								100.00

Table 6.4: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 4 in Random delay.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL	Generative
Fast_departure	0	2739	4187	4473	3909	2756	1093	650
Conservative	1670	0	4001	4358	3780	2549	980	517
Cautious	787	936	0	3636	2958	1819	564	253
Stable	527	642	1363	0	1977	1087	361	114
Optimistic	1091	1220	2042	3022	0	1140	427	136
Greedy	2244	2451	3181	3913	3801	0	940	412
RL	3907	4020	4436	4639	4573	4060	0	1204
Generative	4350	4483	4747	4886	4864	4588	3796	0

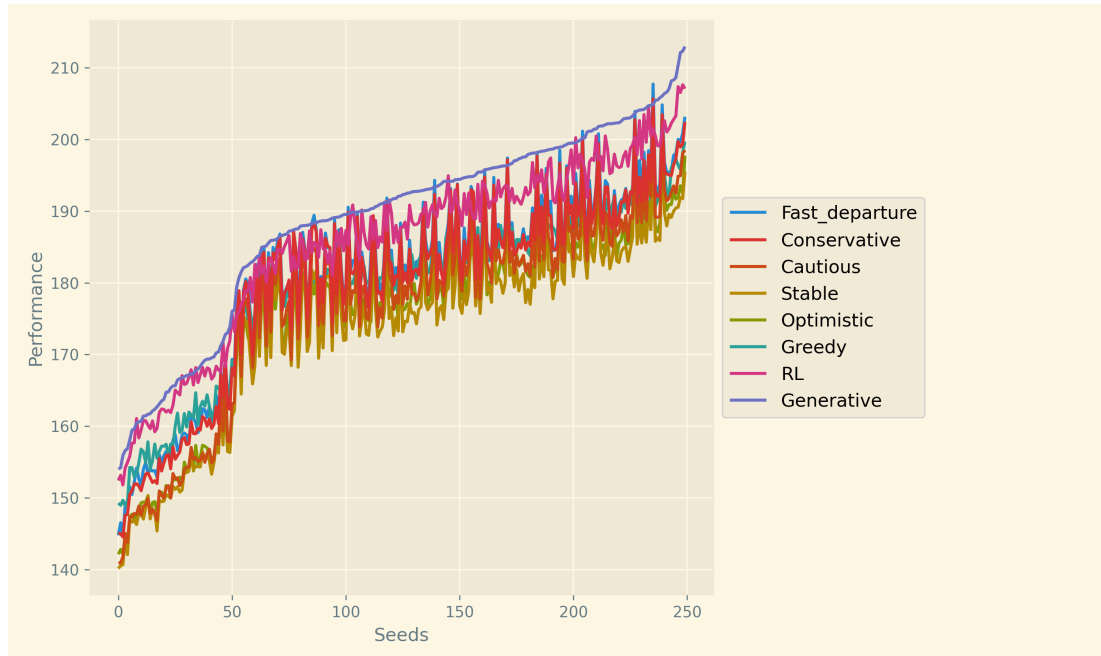


Figure 6.2: Plotting performance of different baselines for Setting 4 in Random delay window size = 20

Performance Analysis for Simulation Setting 4 in Unreliable service provider

Table 6.5: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 4, Unreliable service provider.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL	Generative
Fast_departure	100.00	100.46	102.12	103.49	102.48	99.93	96.81	94.26
Conservative		100.00	101.66	103.01	102.01	99.47	96.37	93.83
Cautious			100.00	101.34	100.35	97.85	94.80	92.30
Stable				100.00	99.03	96.56	93.55	91.08
Optimistic					100.00	97.51	94.46	91.97
Greedy						100.00	96.88	94.33
RL							100.00	97.36
Generative								100.00

Table 6.6: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 4 in Unreliable service provider.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL	Generative
Fast_departure	0	2978	4320	4364	3608	2330	1156	482
Conservative	1394	0	4176	4284	3456	2157	1066	428
Cautious	657	756	0	3599	2648	1543	746	264
Stable	636	716	1400	0	1721	960	482	173
Optimistic	1392	1544	2351	3278	0	976	476	185
Greedy	2670	2843	3457	4040	3951	0	820	357
RL	3844	3934	4254	4518	4524	4175	0	484
Generative	4518	4572	4736	4827	4815	4643	4516	0

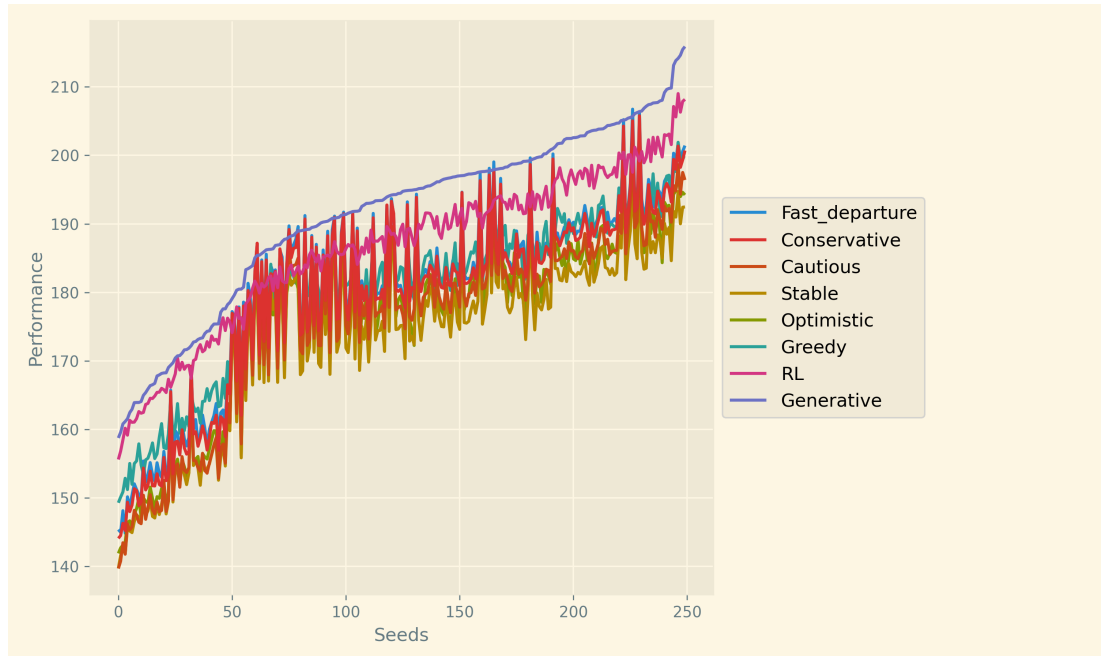


Figure 6.3: Plotting performance of different baselines for Setting 4 in Unreliable service provider window size = 20

Performance Analysis for Simulation Setting 5 in On-time delivery

Table 6.7: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 5, On-time delivery.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL	Generative
Fast_departure	100.00	98.03	94.01	92.31	92.06	92.69	90.86	88.94
Conservative		100.00	95.90	94.17	93.91	94.56	92.69	90.73
Cautious			100.00	98.19	97.93	98.60	96.65	94.61
Stable				100.00	99.73	100.42	98.43	96.35
Optimistic					100.00	100.69	98.70	96.62
Greedy						100.00	98.02	95.95
RL							100.00	97.89
Generative								100.00

Table 6.8: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 5 in On-time delivery.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL	Generative
Fast_departure	0	257	93	152	317	573	271	180
Conservative	4728	0	242	335	546	848	447	298
Cautious	4907	4758	0	1090	1352	1712	867	496
Stable	4848	4665	3909	0	2160	2529	1239	619
Optimistic	4683	4454	3646	2793	0	2568	1257	606
Greedy	4427	4152	3288	2432	1752	0	844	370
RL	4729	4553	4133	3760	3742	4155	0	766
Generative	4815	4697	4503	4381	4394	4630	4233	0

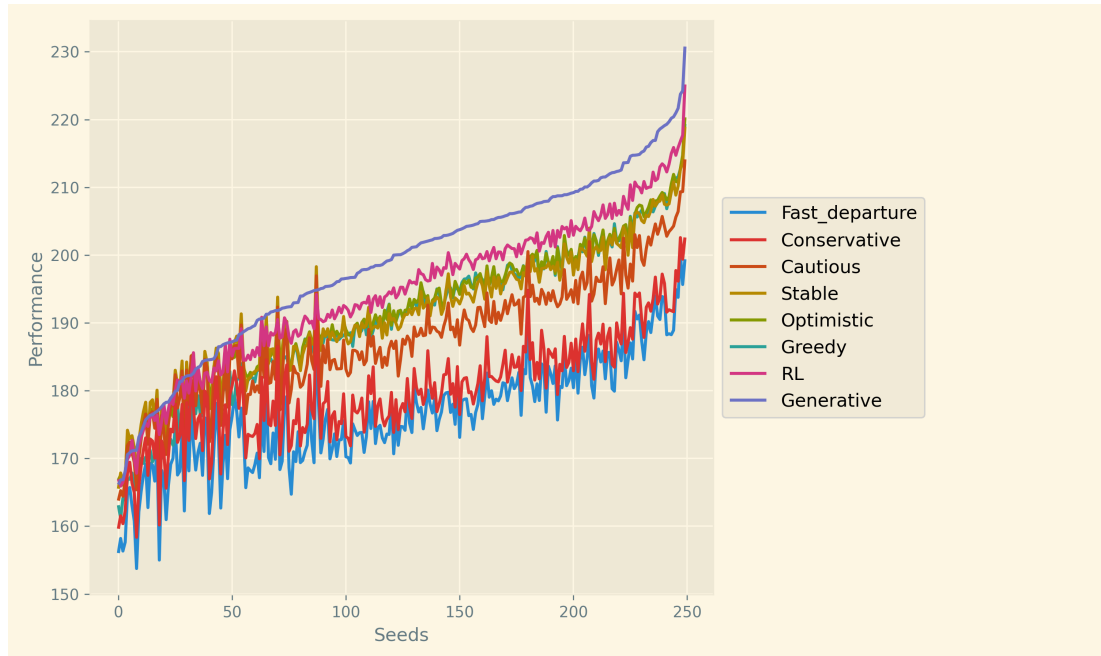


Figure 6.4: Plotting performance of different baselines for Setting 5 in On-time delivery window size = 20

Performance Analysis for Simulation Setting 5 in Random delay

Table 6.9: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 5, Random delay.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL	Generative
Fast_departure	100.00	98.25	94.85	93.67	93.78	94.64	92.69	90.94
Conservative		100.00	96.54	95.34	95.46	96.33	94.34	92.56
Cautious			100.00	98.75	98.87	99.77	97.72	95.87
Stable				100.00	100.12	101.03	98.95	97.08
Optimistic					100.00	100.91	98.83	96.96
Greedy						100.00	97.94	96.09
RL							100.00	98.11
Generative								100.00

Table 6.10: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 5 in Random delay.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL	Generative
Fast_departure	0	712	219	324	553	908	370	267
Conservative	4278	0	505	584	870	1227	611	411
Cautious	4781	4493	0	1589	1874	2225	1299	727
Stable	4676	4415	3409	0	2571	2950	1761	859
Optimistic	4446	4130	3125	2393	0	2727	1660	714
Greedy	4092	3773	2773	2017	1599	0	1198	348
RL	4629	4388	3698	3236	3337	3800	0	789
Generative	4733	4589	4273	4141	4286	4652	4209	0

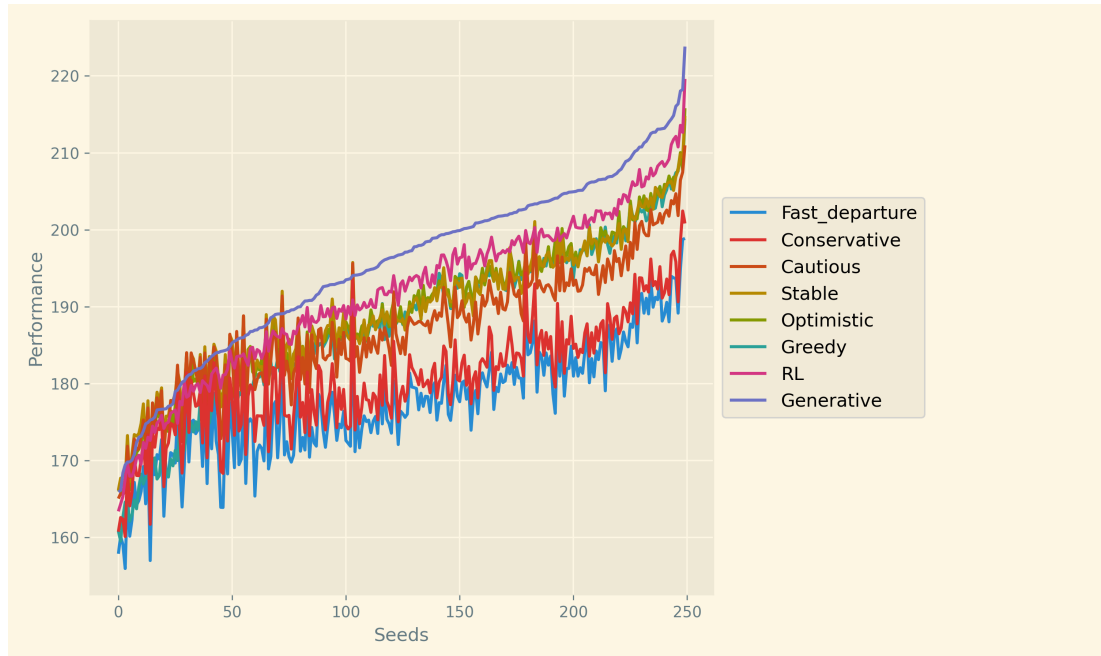


Figure 6.5: Plotting performance of different baselines for Setting 5 in Random delay window size = 20

Performance Analysis for Simulation Setting 5 in Unreliable service provider

Table 6.11: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 5, Unreliable service provider.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL	Generative
Fast_departure	100.00	98.16	94.58	93.14	93.12	93.91	91.82	89.89
Conservative		100.00	96.35	94.88	94.87	95.67	93.55	91.57
Cautious			100.00	98.47	98.46	99.29	97.09	95.04
Stable				100.00	99.98	100.83	98.59	96.51
Optimistic					100.00	100.85	98.61	96.53
Greedy						100.00	97.78	95.72
RL							100.00	97.89
Generative								100.00

Table 6.12: Table of seed counts where the row baseline strictly outperforms the column baseline for Setting 5 in Unreliable service provider.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL	Generative
Fast_departure	0	382	215	351	574	869	414	250
Conservative	4603	0	404	577	841	1175	616	383
Cautious	4785	4595	0	1358	1665	2033	1071	609
Stable	4649	4423	3639	0	2418	2771	1453	780
Optimistic	4426	4159	3333	2540	0	2739	1384	744
Greedy	4131	3824	2967	2187	1573	0	948	460
RL	4584	4383	3927	3545	3614	4052	0	667
Generative	4745	4613	4391	4220	4255	4540	4333	0

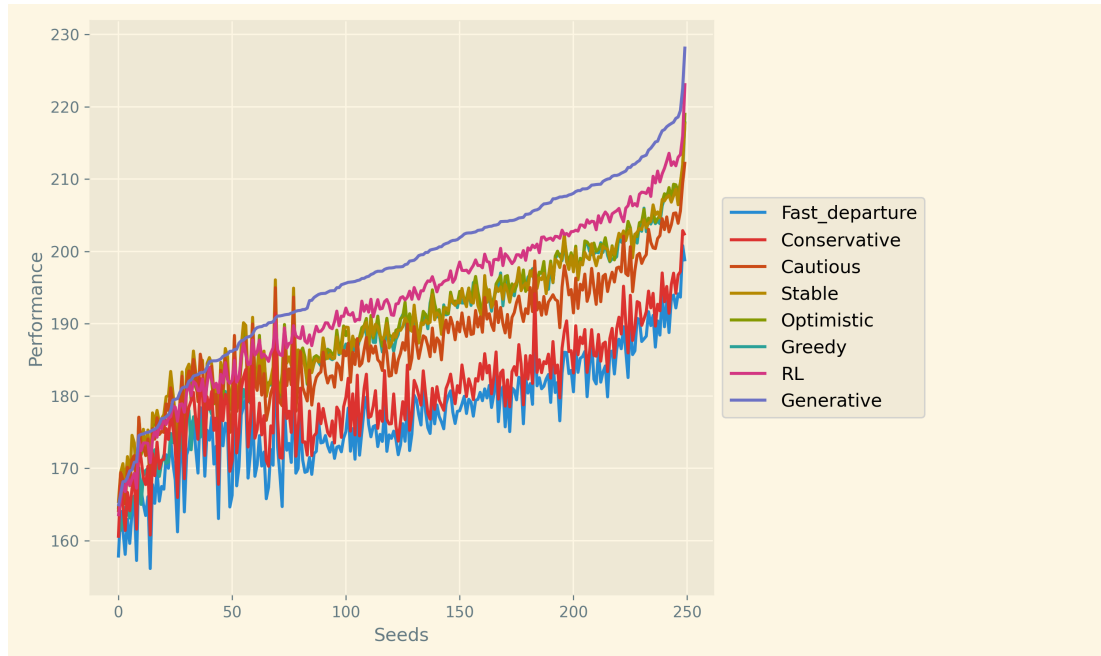


Figure 6.6: Plotting performance of different baselines for Setting 5 in Unreliable service provider window size = 20

6.4 Discussion

The results presented in this chapter provide a clear case for the superiority of the GRM agent in settings 4 and 5. While the preceding pairwise comparisons hinted at the GRM agent's advantage, the data in Table 6.13 offers proof of decision-making capabilities. In Setting 4, under the "Unreliable service provider" scenario, the RL agent wins 3,344 times (66.9%), leaving over 33% of the runs to be won by simpler strategies. In contrast, the GRM agent achieves the top score in **4,218** runs (84.4%) under the same conditions. In Setting 5 and under the most challenging "Random delay" scenario, RL agent was able to secure the top score in **2,441** times out of 5000 or (48.8%), meaning it fails to find the optimal strategy in more than half of the simulations. This is where the GRM agent achieves the maximum score **3,719** times (74.4%).

Mechanism of Success The superior performance documented in Table 6.13 is directly attributable to the GRM's architecture. First, the packet arrival estimator provides the agent with a stable, high-level context, preventing it from being misled by noisy, short-term fluctuations. Second, the reward-averaging mechanism filters out the stochasticity of individual outcomes, giving the agent a more reliable learning signal that points toward long-term, robust strategies. By effectively tackling both state uncertainty and reward variance, the GRM agent learns a policy that is high-performing on average.

Table 6.13: Comparative Performance of RL and Generative Agents Against Baselines. The table shows the frequency (out of 5000 runs) that each strategy achieved the maximum performance score.

Scenario and Setting	Fast_departure	Greedy	Optimistic	Stable	Cautious	Conservative	RL	Generative
On-time delivery								
Setting 4 RL	597	369	46	18	33	227	3803	-
Setting 4 GRM	258	168	12	2	11	86	-	4508
Setting 5 RL	9	471	536	627	426	82	2947	-
Setting 5 GRM	5	194	240	277	219	51	-	4041
Random delay								
Setting 4 RL	681	699	101	46	94	425	3104	-
Setting 4 GRM	456	327	49	22	38	247	-	3936
Setting 5 RL	20	505	659	814	556	137	2441	-
Setting 5 GRM	10	164	267	421	342	105	-	3719
Unreliable service provider								
Setting 4 RL	755	503	75	33	84	368	3344	-
Setting 4 GRM	309	274	43	19	35	164	-	4218
Setting 5 RL	48	454	565	705	476	146	2726	-
Setting 5 GRM	19	199	271	355	238	78	-	3884

6.5 Conclusion

This chapter set out to address the critical challenge of deploying RL agents in the Settings 4 and 5 in GymDC environment. The investigation centered on a novel GRM-based agent designed specifically to handle environmental noise and non-stationarity. The GRM agent outperformed both the RL agent and all heuristic baselines across these settings. This success is rooted in the GRM's capacity to abstract away environmental noise and learn from a stabilized reward signal. In conclusion, this work validates the GRM as a framework for developing autonomous decision-making systems for logistical challenges.

Chapter 7

Concluding remarks

This thesis established a framework for autonomous decision-making within the complex and stochastic environment of a PI distribution center hub. The central claim, that RL can be utilized to learn optimal policies for DC control and other examples characterized by poor observability, was validated through a combination of custom-built simulation environments and empirical validation in multiple competition domains. In summary, this thesis makes several key contributions:

Gym-DC Simulation Environment: The creation of Gym-DC provides the first publicly available RL benchmark for PI DC hub, modeling packet and service provider variability, storage constraints, and stochastic delivery scenarios. Empirical results showed that RL agent outperforms basic heuristic policies in low-complexity settings, successfully validating the first part of the thesis statement.

Theoretical Classification Framework: A novel framework for classifying RL problem complexity was introduced, founded on the formal existence of optimal policies. This framework is underpinned by a theorem demonstrating that under specific conditions of non-stationary rewards dominated by aleatoric uncertainty, an optimal policy does not exist. This theoretical insight guides the selection and design of advanced RL architectures for the hard problems.

The Generative Reward Machine Framework: GRM was introduced as an extension of the traditional Reward Machine. The GRM addresses fundamental limitations of conventional RL by generating intermediate abstract state representations that effectively transform highly stochastic and partially observable environments into tractable learning problems.

Gym-DC Hard Settings: The GRM agent outperform the standard RL agent and all heuristics, securing the maximum score.

Competitive Domains: The GRM agent achieved first place in both the International StepUP Competition for Biometric Footstep Recognition and the ECML Discovery Challenge on Foundation Models for Genomic Modeling and Understanding. The Bot, also using the GRM framework, performed competitively at the NeurIPS 2022 Tournament, proving the GRM's effectiveness in adaptive policy refinement in imperfect information games.

7.1 Future work

Future work will focus on three key areas: extending the simulation to a multi-agent setting, enhancing the Physical Internet PI simulation, and improving the Generative Reward Machine.

7.1.1 Future Work in GymDC

The GymDC environment should be extended to a Multi-Agent Reinforcement Learning environment where multiple autonomous DC hubs and transporters act independently. The primary challenge here will be developing cooperative and competitive policies for fully decentralized routing and dynamic capacity negotiation in a multi-criteria, partially observable setting. This directly aligns with the broader vision of the fully realized Physical Internet.

Regarding the DC hub, the current scenario simulates one DC. We have considered three different scenarios for packet delivery: "On-time delivery", "Random delay" and "Unreliable service provider". One research direction is to extend the simulator to consider several hubs with different policies, which may cause delays in delivery. In this case, the agent in the hub will face another source of noise and will need a policy that works in this situation. This scenario can be extended to a multi-agent setting, where agents may be trained in an environment where they consider the side effects of each other during training. Researchers may try to implement multi-agent scenarios and then evaluate RL algorithms in these scenarios.

7.1.2 Future Work in PI Simulation

Two other areas that are good directions to implement with a simulator and RL environment are:

Transport Optimization

The performance of π – Movers, such as trucks, trains, and ships, which represent the Physical and Link Layers of the PI, can be improved by RL. Simulators can model the movement and cooperation of these vehicles, allowing RL agents to learn dynamic coordination strategies. Specifically, RL can reduce empty backhaul runs and minimize fuel consumption by dynamically matching empty capacity with available π – containers across the network in real-time. For road transport, RL agents can control the formation and dynamic reconfiguration of truck platoons at virtual transfer points to maximize fuel efficiency and vehicle utilization, especially when dealing with high-volume flows. This shift from static scheduling to adaptive, cooperative logistics is essential for lowering the total logistical cost and achieving PI's sustainability goals.

Protocols and Management

Simulation platforms incorporating RL can be used to test the stability and economic viability of various PI Governance and business protocols. RL agents, representing individual Logistics Service Providers, can learn and optimize their cooperation strategies within new models for shared capacity. This would involve demonstrating the profitability of collaboration and testing mechanisms that incentivize trust and data sharing.

7.1.3 Future Work in Enhancing the GRM

The current GRM relies on either a pre-trained estimator or estimators. Future work should focus on integrating this estimation directly into the online learning loop:

Joint Online Learning of Estimator and Policy

Develop an end to end framework where the state-abstraction mechanism the estimator and the action-selection policy are learned concurrently. This would allow the agent to continuously adapt its understanding of the environment's complexity, eliminating the need for offline data generation and clustering.

Reward Function Synthesis

Investigate methods to dynamically synthesize the abstract reward function for the estimated state space, moving beyond the current simple reward averaging. This could involve deep generative models to predict the optimal stationary reward signal, further enhancing sample efficiency and learning stability in hard problems.

By addressing these aspects and conducting additional experiments, researchers may be able to improve RL performance in current examples of hard-class complexity in reinforcement learning environments.

References

- [AHTA17] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 22–31. PMLR, 06–11 Aug 2017.
- [ALI17] ALICE. Alliance for logistics innovation through collaboration in europe, 2017.
- [Alt04] Eitan Altman. *Constrained Markov decision processes*. Publisher: Route des Lucioles, B.P.9306902 Sophia-Antipolis Cedex France, 2004.
- [AMGW⁺19] Jose A Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. Rudder: Return decomposition for delayed rewards. *Advances in Neural Information Processing Systems*, 32, 2019.
- [APH⁺21] Moloud Abdar, Farhad Pourpanah, Sadiq Hussain, Dana Rezazadegan, Li Liu, Mohammad Ghavamzadeh, Paul Fieguth, Xiaochun Cao, Abbas Khosravi, U. Rajendra Acharya, Vladimir Makarenkov, and Saeid Nahavandi. A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Information Fusion*, 76:243–297, 2021.
- [BCP⁺16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.
- [BDG⁺21] Sam Ban, Andrea Dan, Félix Guinet, Julien Portugaunel, Sarot Srang, and Matthieu Luras. Assessing the potentialities of

- physical internet for developing countries last mile deliveries. In *8th International Physical Internet Conference*, 2021.
- [Bel57] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, 6(5):679–684, 1957.
- [Bis06] Johannes Bisschop. *AIMMS - Optimization Modeling*. Lulu.com , ISBN-13 : 978-1847539120, 2006.
- [BM20] Jana Boerger and Benoit Montreuil. Data-driven analytics-based capacity management for hyperconnected third-party logistics providers. In *7th International Physical Internet Conference*, Shenzhen, China, 2020.
- [BS18] Noam Brown and Tuomas Sandholm. Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.
- [BS19] Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365(6456):885–890, 2019.
- [CAS16] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198, 2016.
- [CBRT22] Tarik Chargui, Abdelghani Bekrar, Mohamed Reghioui, and Damien Trentesaux. Scheduling trucks and storage operations in a multiple-door cross-docking terminal considering multiple storage zones. *International Journal of Production Research*, 60(4):1153–1177, 2022.
- [CCML18] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*, pages 4754–4765, 2018.
- [CCPLR21] David Cipres, Alberto Capella, Lorena Polo, and Jose L. Lopez Ramon. Gpics: a framework to create a digital twin for the physical internet. In *8th International Physical Internet Conference*, Zaragoza, Spain, 2021.

- [CDM⁺21] Miguel Campos, Shahab Derhami, Leon McGinnis, Benoit Montreuil, and Ali V. Barenji. Digital twin design requirements for durable goods distribution in physical internet. In *8th International Physical Internet Conference*, 2021.
- [Cea19] Alberto Camacho and et al. Ltl and beyond: Formal languages for reward function specification in reinforcement learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*. IJCAI, 2019.
- [Cha25] Challenge Organizers. Ecml discovery challenge on foundation models for genomic modeling and understanding. <https://www.codabench.org/competitions/6930/>, September 2025. Accessed October 19, 2025.
- [CHJH02] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.
- [Cla21] Gregory Clark. Deep synoptic monte carlo planning in reconnaissance blind chess. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS ’21, Red Hook, NY, USA, 2021. Curran Associates Inc.
- [DAEvH⁺15] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015.
- [DALM⁺21] Gabriel Dulac-Arnold, N. Levine, Daniel J. Mankowitz, et al. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110:2419–2468, 2021.
- [DDV⁺18] Gal Dalal, Krishnamurthy Dvijotham, Mel Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*, 2018.
- [DF21] Chuanwen Dong and Rod Franklin. From the digital internet to the physical internet: A conceptual framework with a stylized

- network model. *Journal of Business Logistics*, 42(1):108–119, 2021.
- [DHS20] Igor Y. Davydenko, W.M.M. Hopman, and J.S. Sreen. Decentralized and centralized transport and logistics carbon emission optimization and emission norms for the transport and logistics sector. In *Proceedings of the 7th International Physical Internet Conference*, Shenzhen, China, 2020.
- [FAL17] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *Proceedings of the 34th International Conference on Machine Learning*, 2017.
- [FID22] FIDE. FIDE & Google announce sponsorship agreement and new AI Challenge. <https://www.fide.com/news/1802>, 2022. Accessed: June 2024.
- [FMP19] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*, pages 2052–2062. PMLR, 2019.
- [FSLvM23] Ruben Fransen, Lola Sprenger, Wan-Jui Lee, and Jaco van Meijeren. Performance of self-organizing logistics: a practical comparison between centralized and decentralized logistics. In *Proceedings of the 9th International Physical Internet Conference (IPIC 2023)*, Athens, Greece, June 2023.
- [FTP23] Geoffrey Featherstone, Russell G. Thompson, and Medo Pournader. Asset capability, transparency, and governance - fundamental to pi proliferation. In *Proceedings of the 9th International Physical Internet Conference*, Athens, Greece, 2023.
- [Fé25] Fédération Internationale des Echecs (FIDE). Fide google efficient chess ai challenge. In *Official Website*, Online, 2025. The kaggle Team. Available at: <https://www.kaggle.com/competitions/fide-google-efficiency-chess-ai-challenge>. Accessed on March 1, 2025.
- [GBCB16] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.
- [GLR⁺20] Ryan W. Gardner, Corey Lowman, Casey Richardson, Ashley J. Llorens, Jared Markowitz, Nathan Drenkow, Andrew Newman,

- Gregory Clark, Gino Perrotta, Robert Perrotta, Timothy Highley, Vlad Shcherbina, William Bernatoni, Mark Jordan, and Asen Asenov. The first international competition in machine reconnaissance blind chess. In *Proceedings of the NeurIPS 2019 Competition and Demonstration Track*, volume 123 of *Proceedings of Machine Learning Research*, pages 121–130. PMLR, 2020.
- [GPD21] Lukas Gosch, Matthias Prandstetter, and Karl F. Doerner. On modelling and solving green collaborative tactical transportation planning. In *Proceedings of the 8th International Physical Internet Conference (IPIC 2021)*, 2021.
- [GPS⁺22] Ryan W. Gardner, Gino Perrotta, Anvay Shah, Shivaram Kalyanakrishnan, Kevin A. Wang, Gregory Clark, Timo Bertram, Johannes Fürnkranz, Martin Müller, Brady P. Garrison, Prithviraj Dasgupta, and Saeid Rezaei. The machine reconnaissance blind chess tournament of neurips 2022. In *Proceedings of the NeurIPS 2022 Competitions Track*, volume 220 of *Proceedings of Machine Learning Research*, pages 119–132. PMLR, 2022.
- [GSBS25] Naruphon Gunhawan, Apichat Sopadang, Ruth Banomyong, and Salinee Santiteerakul. Readiness indicators for physical internet in supply chain. In *Proceedings of the 11th International Physical Internet Conference*, Hong Kong, China, 2025.
- [hb21] <https://stablebaselines3.readthedocs.io/en/master/misc/projects.html>. Stable baselines3. <https://github.com/DLR-RM/stable-baselines3?tab=readme-ov-file#stable-baselines3>, 2021.
- [Hof23] Wout Hofman. Pi data sharing infrastructure. In *Proceedings of the 9th International Physical Internet Conference*, Athens, Greece, 2023.
- [HS13] Todd Hester and Peter Stone. Texlore: Real-time sample-efficient reinforcement learning for robots. *Machine learning*, 90(3):385–429, 2013.
- [HS15] Matthew J Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *AAAI fall symposia*, volume 45, page 141, 2015.

- [HVP⁺18] Todd Hester, Mel Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.
- [HW21] Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods. *Machine Learning*, 110:457–506, 2021.
- [IKVM22] Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research*, 73:173–208, 2022.
- [Int19] International Transport Forum. Itf transport outlook 2019. Technical report, OECD, Paris, 2019. Available at <https://www.itf-oecd.org/sites/default/files/docs/itf-transport-outlook-2019.pdf>.
- [JS11] Nathalie Japkowicz and Mohak Shah. *Evaluating learning algorithms: a classification perspective*. Cambridge University Press, 2011.
- [KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [KL19] Justin Fu Matthew Soh George Tucker Kumar, Aviral and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, 2019.
- [KLF20] Steffen Kaup, André Ludwig, and Bogdan Franczyk. Design and evaluation of routing artifacts as a part of the physical internet framework. In *Physical Internet: Proceedings of the 7th International Physical Internet Conference, IPIC 2020*, 2020.
- [KLF21] Steffen Kaup, André Ludwig, and Bogdan Franczyk. Framework artifact for the road-based physical internet based on internet protocols. In *Proceedings of the 8th International Physical Internet Conference (IPIC 2021)*, 2021.
- [KM21] Sara Kaboudvand and Benoit Montreuil. Dynamic containerized

- consolidation in physical internet enabled parcel logistics. In *Proceedings of the 8th International Physical Internet Conference*, 2021.
- [KMKK21] Nayeon Kim, Benoit Montreuil, Walid Klibi, and Nitish Kholgade. Hyperconnected urban fulfillment and delivery. *Transportation Research Part E: Logistics and Transportation Review*, 145:102104, 2021.
- [KT23] Cevahir Koprulu and Ufuk Topcu. Reward-machine-guided, self-paced reinforcement learning. In *Uncertainty in Artificial Intelligence*, pages 1121–1131. PMLR, 2023.
- [LCK⁺22] Andrew C. Li, Zizhao Chen, Toryn Q. Klassen, Pashootan Vaezipoor, Rodrigo Toro Icarte, and Sheila A. McIlraith. Reward machines for deep rl in noisy and uncertain environments. In *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*, pages 23377–23390, 2022.
- [LKTF20] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, survey, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.
- [LMP⁺25] Robyn Larracy, Eve MacDonald, Angkoon Phinyomark, Saeid Rezaei, Mahdi Laghaei, Ali Hajighasem, Aaron Tabor, and Erik Scheme. First international stepup competition for biometric footstep recognition: Methods, results and remaining challenges. September 2025.
- [LPS⁺25] Robyn Larracy, Angkoon Phinyomark, Ala Salehi, Eve MacDonald, Saeed Kazemi, Shikder Shafiul Bashar, Aaron Tabor, and Erik Scheme. Unb stepup: A footstep database for gait analysis and recognition using underfoot pressure. *arXiv preprint arXiv:2502.17244*, 2025.
- [MBF12] B. Montreuil, E. Ballot, and F. Fontane. An open logistics interconnection model for the physical internet. In *Proceedings of the 14th IFAC Symposium on Information Control Problems in Manufacturing (INCOM 2012)*, pages 327–332, Bucharest, Romania, 2012.
- [MH18] Sultan Javed Majeed and Marcus Hutter. On q-learning

- convergence for non-markov decision processes. *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)*, 2018.
- [MHM18] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.
- [MKS⁺15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.
- [MLM⁺23] Marco Mussi, Davide Lombarda, Alberto Maria Metelli, Francesco Trovò, and Marcello Restelli. Arlo: A framework for automated reinforcement learning. *Expert Systems with Applications*, 224:119883, 2023.
- [MMB13] Benoit Montreuil, Russell D Meller, and Eric Ballot. Physical internet foundations. In *Service orientation in holonic and multi agent manufacturing and robotics*, pages 151–166. Springer, 2013.
- [MMC⁺04] Dario Maio, Davide Maltoni, Raffaele Cappelli, James L. Wayman, and Anil K. Jain. Fvc2004: Third fingerprint verification competition. In *International Conference on Biometric Authentication*, 2004.
- [Mon11] Benoît Montreuil. Toward a physical internet: meeting the global logistics sustainability grand challenge. *Logistics Research*, 3:71–87, 2011.
- [MRN05] Francisco A Melo, M Isabel Ribeiro, and ISR Torre Norte. Convergence results for reinforcement learning with partial observability. *Institute for Systems and Robotics, Tech. Rep. RT-602-05*, 2005.
- [MW02] A. J. Mansfield and J. L. Wayman. Best practices in testing and reporting performance of biometric devices. Technical Report CMSC 14/02, National Physical Laboratory, August 2002. Version 2.01.

- [NFS25] Gero Niemann, Rod Franklin, and Malte Spanuth. Addressing the reachability problem in the physical internet: A pi-link protocol inspired by the internet. In *Proceedings of the 11th International Physical Internet Conference (IPIC 2025)*, Hong Kong, China, June 2025.
- [NRK⁺16] Andrew J. Newman, Casey L. Richardson, Sean M. Kain, Paul G. Stankiewicz, Paul R. Guseman, Blake A. Schreurs, and Jeffrey A. Dunne. Reconnaissance blind multi-chess: An experimentation platform for ISR sensor fusion and resource management. In Ivan Kadar, editor, *Signal Processing, Sensor/Information Fusion, and Target Recognition XXV*, volume 9842, page 984209. SPIE, 2016.
- [ope21] openai. gym-cartpole-swingup. <https://github.com/0xangelo/gym-cartpole-swingup>, 2021.
- [PAZA18] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE, 2018.
- [PBHM17] Shenle Pan, Eric Ballot, George Q Huang, and Benoit Montreuil. Physical internet and interconnected logistics services: research and applications. *International Journal of Production Research*, pages 2603–2609, 2017.
- [PGM⁺19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8026–8037, 2019.
- [PHRS⁺22] Jack Parker-Holder, Raghu Rajan, Xingyou Song, André Biedenkapp, Yingjie Miao, Theresa Eimer, Baohe Zhang, Vu Nguyen, Roberto Calandra, Aleksandra Faust, et al. Automated reinforcement learning (autorl): A survey and open

- problems. *Journal of Artificial Intelligence Research*, 74:517–568, 2022.
- [PKD23] Vinay Pandey, Komal, and Hasan Dincer. A review on topsis method and its extensions for different applications with recent development. *Soft computing in decision making and in modeling in economics*, 27:18011–18039, 2023.
- [Pow22] Warren B Powell. *Reinforcement learning and stochastic optimization: A unified framework*. John Wiley & Sons, 2022.
- [Put14] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.
- [PyT25] PyTorch Documentation. L1loss. <https://docs.pytorch.org/docs/>, October 2025. PyTorch Documentation.
- [QSZ⁺20] Wei Qin, Yanning Sun, Zilong Zhuang, Zhiyao Lu, and Yaoming Zhou. Multi-agent reinforcement learning-based dynamic task assignment for vehicles in urban transportation physical internet. In *Proceedings of the 7th International Physical Internet Conference, IPIC 2020*, 2020.
- [RCG⁺15] Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.
- [Rud16] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016.
- [RVWD13] Diederik M Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113, 2013.
- [SB18] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.
- [SBHK19] Georg Schett, Georg Brunthaller, Christina Hess, and Stefanie Kritzinger. Conceptual model of a decentralized transport organization in the increasingly uncertain transport

- environment of the physical internet. In *Proceedings of the 6th International Physical Internet Conference (IPIC 2019)*, 2019.
- [Sha50] Claude E Shannon. Xxii. programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 41(314):256–275, 1950.
- [SHM⁺16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.
- [SHS⁺17] David Silver, Thomas Hubert, Julian Schrittwieser, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017.
- [SHS⁺18] David Silver, Thomas Hubert, Julian Schrittwieser, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.
- [SK19] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.
- [SLH⁺14] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr, 2014.
- [SM24] Sahrish Shaikh and Benoit Montreuil. Dynamic directional routing for the physical internet: A sector-based approach with dynamic adjustment. In *Proceedings of the 10th International Physical Internet Conference*, Savannah, GA, USA, 2024.
- [SMM⁺24] Shiqi Sun, Philippe Michiels, Cathy Macharis, An Cant, Dries Van Bever, and Koen Mommens. Unlocking the potential of the physical internet: a trust-enabling decentralized process sharing connector. In *Proceedings of the 10th International Physical Internet Conference*, Savannah, GA, USA, 2024.
- [SNS⁺19] Wolfgang Schildorfer, Matthias Neubauer, Oliver Schauer,

- Andreas Kuhr, and Walter Aigner. Revealing mutual relationships between truck platooning and smart hyperconnected physical internet systems. In *Proceedings of the 6th International Physical Internet Conference (IPIC 2019)*, London, UK, July 2019.
- [The25] The Stockfish Team. Stockfish - open source chess engine. In *Official Website*, Online, 2025. The Stockfish Team. Available at: <https://stockfishchess.org/>. Accessed on June 26, 2022.
- [TIKVM18] Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *Proceedings of the International Conference on Machine Learning*. PMLR, 2018.
- [TMLZ20] Horst Treiblmaier, Kristijan Mirkovski, Paul Benjamin Lowry, and Zach G. Zacharia. The physical internet as a new supply chain paradigm: a systematic literature review and a comprehensive framework. *The International Journal of Logistics Management*, 2020.
- [VSFR⁺17] Harm Van Seijen, Mehdi Fatemi, Joshua Romoff, Romain Laroche, Tavian Barnes, and Jeffrey Tsang. Hybrid reward architecture for reinforcement learning. *Advances in neural information processing systems*, 30, 2017.
- [VTN23] Anshul Vijay, Russell G. Thompson, and Neema Nassir. A proposal and evaluation of a digital twin framework for PI-Hubs using re-enforcement learning based multi-agent systems model. In *Proceedings of the 9th International Physical Internet Conference (IPIC 2023)*, Athens, Greece, June 2023.
- [WOL⁺23] Ziqing Wu, Rapha"el Oger, Matthieu Luras, Benoit Montreuil, and Louis Faug"ere. Physical internet enabled hyperconnected circular supply chains. In *Proceedings of the 9th International Physical Internet Conference*, Athens, Greece, 2023.
- [YL25] Heng Yang and Ke Li. ECML discovery challenge on foundation models for genomic modeling and understanding. <https://www.codabench.org/competitions/6930/>, 2025. Part of the European Conference on Machine Learning and Principles

and Practice of Knowledge Discovery in Databases (ECML PKDD) 2025.

Chapter 8

Appendices

This appendix presents a table for each setting within each scenario. These tables display the **Selective Performance Comparison Matrix**, where each cell $[i][j]$ represents the ratio of the mean performance value in cases where baseline i outperforms baseline j across 5000 simulation runs. This approach highlights instances where strategy i demonstrates superior performance over strategy j at corresponding indices. The caption of each table specifies the simulation setting and scenario.

- section 8.1 presents tables comparing only the baseline strategies for settings 1 to 5.
- section 8.2 includes tables comparing baseline strategies and the RL agent for settings 1, 2, and 3.
- section 8.3 contains tables comparing baseline strategies, the RL agent, and the GRL agent for settings 4 and 5.

8.1 Comparative Performance of Baseline Strategies

Table 8.1: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 1, On-time delivery.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	101.05	104.12	0	0	0
Conservative		0	102.27	109.22	0	0
Cautious			0	102.69	101.88	0
Stable				0	104.27	0
Optimistic					0	0
Greedy						0

Table 8.2: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 2, On-time delivery.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	100.60	101.82	101.93	101.93	101.93
Conservative		0	102.29	102.48	102.48	102.48
Cautious			0	101.07	101.07	101.07
Stable				0	0	0
Optimistic					0	0
Greedy						0

Table 8.3: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 3, On-time delivery.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	107.91	107.91	107.91	107.91	107.91
Conservative		0	101.74	101.74	101.74	101.74
Cautious			0	0	0	0
Stable				0	0	0
Optimistic					0	0
Greedy						0

Table 8.4: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 4, On-time delivery.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	101.00	102.42	103.80	103.84	103.42
Conservative		0	101.98	103.38	103.47	103.17
Cautious			0	102.09	102.46	102.50
Stable				0	101.40	101.78
Optimistic					0	101.14
Greedy						0

Table 8.5: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 5, On-time delivery.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	100.44	100.97	101.43	102.05	103.95
Conservative		0	100.95	101.47	102.28	104.01
Cautious			0	101.22	101.98	103.38
Stable				0	101.40	102.65
Optimistic					0	101.97
Greedy						0

Table 8.6: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 1, Random delay.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	104.14	105.29	106.47	104.73	104.80
Conservative		0	104.68	104.78	103.30	102.44
Cautious			0	103.08	101.13	100.80
Stable				0	101.44	100.87
Optimistic					0	101.03
Greedy						0

Table 8.7: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 2, Random delay.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	100.70	102.00	102.10	102.10	102.10
Conservative		0	102.50	102.68	102.68	102.68
Cautious			0	101.43	101.43	101.43
Stable				0	0	0
Optimistic					0	0
Greedy						0

Table 8.8: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 3, Random delay.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	108.22	108.22	108.22	108.22	108.22
Conservative		0	102.06	102.06	102.06	102.06
Cautious			0	0	0	0
Stable				0	0	0
Optimistic					0	0
Greedy						0

Table 8.9: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 4, Random delay.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	101.43	102.85	104.37	104.51	104.01
Conservative		0	102.42	103.96	104.11	103.78
Cautious			0	102.53	102.97	102.98
Stable				0	101.79	102.27
Optimistic					0	101.49
Greedy						0

Table 8.10: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 5, Random delay.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	100.76	101.04	101.50	102.23	103.95
Conservative		0	101.10	101.67	102.46	104.21
Cautious			0	101.50	102.27	103.71
Stable				0	101.76	103.03
Optimistic					0	102.36
Greedy						0

Table 8.11: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 1, Unreliable service provider.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	103.41	107.89	111.03	114.23	113.50
Conservative		0	106.75	110.85	112.69	112.30
Cautious			0	107.34	109.86	109.55
Stable				0	108.97	111.18
Optimistic					0	114.64
Greedy						0

Table 8.12: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 2, Unreliable service provider.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	102.81	102.52	102.61	102.61	102.61
Conservative		0	102.46	102.65	102.65	102.65
Cautious			0	101.11	101.11	101.11
Stable				0	0	0
Optimistic					0	0
Greedy						0

Table 8.13: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 3, Unreliable service provider.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	108.44	108.45	108.45	108.45	108.45
Conservative		0	102.51	102.51	102.51	102.51
Cautious			0	0	0	0
Stable				0	0	0
Optimistic					0	0
Greedy						0

Table 8.14: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 4, Unreliable service provider.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	101.04	102.56	104.13	104.31	104.19
Conservative		0	102.09	103.67	103.93	103.93
Cautious			0	102.25	102.82	103.18
Stable				0	101.69	102.46
Optimistic					0	101.53
Greedy						0

Table 8.15: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 5, Unreliable service provider.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy
Fast_departure	0	100.51	101.64	102.63	103.41	104.96
Conservative		0	101.38	102.27	103.23	104.80
Cautious			0	101.52	102.47	103.90
Stable				0	101.59	102.98
Optimistic					0	102.10
Greedy						0

8.2 Comparative Analysis of the RL Agent and Baseline Strategies

Table 8.16: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 1, On-time delivery. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.

	Fast _{departure}	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast _{departure}	0	108.26	126.90	149.08	159.49	161.93	161.93
Conservative		0	117.49	137.79	147.41	149.67	149.67
Cautious			0	118.17	126.45	128.38	128.38
Stable				0	113.29	116.05	116.05
Optimistic					0	115.77	115.77
Greedy						0	0
RL							0

Table 8.17: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 1, Random delay. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.

	Fast _{departure}	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast _{departure}	0	107.51	118.84	132.98	139.91	141.60	141.60
Conservative		0	113.18	126.19	132.68	134.27	134.27
Cautious			0	113.30	118.97	120.37	120.37
Stable				0	109.60	111.48	111.48
Optimistic					0	110.52	110.52
Greedy						0	0
RL							0

Table 8.18: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 1, Unreliable service provider. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.

	Fast _{departure}	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast _{departure}	0	107.55	124.03	142.71	151.40	153.50	153.50
Conservative		0	116.03	133.46	141.78	143.61	143.61
Cautious			0	116.61	123.94	125.56	125.56
Stable				0	112.70	115.10	115.10
Optimistic					0	115.09	115.09
Greedy						0	0
RL							0

Table 8.19: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 2, On-time delivery. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.

	Fast _{departure}	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast _{departure}	0	102.57	102.01	101.93	101.93	101.93	103.09
Conservative		0	100.55	100.50	100.50	100.50	100.85
Cautious			0	100.25	100.25	100.25	102.73
Stable				0	0	0	102.92
Optimistic					0	0	102.92
Greedy						0	102.92
RL							0

Table 8.20: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 2, Random delay. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.

	Fast _{departure}	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast _{departure}	0	102.63	102.06	101.99	101.99	101.99	103.49
Conservative		0	100.70	100.66	100.66	100.66	101.48
Cautious			0	100.59	100.59	100.59	103.34
Stable				0	0	0	103.55
Optimistic					0	0	103.55
Greedy						0	103.55
RL							0

Table 8.21: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 2, Unreliable service provider. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.

	Fast _{departure}	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast _{departure}	0	102.93	102.58	102.52	102.52	102.52	104.10
Conservative		0	101.05	101.09	101.09	101.09	101.40
Cautious			0	100.37	100.37	100.37	103.20
Stable				0	0	0	103.42
Optimistic					0	0	103.42
Greedy						0	103.42
RL							0

Table 8.22: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 3, On-time delivery. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.

	Fast _{departure}	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast _{departure}	0	101.33	101.33	101.33	101.33	101.33	101.88
Conservative		0	100.74	100.74	100.74	100.74	109.49
Cautious			0	0	0	0	109.49
Stable				0	0	0	109.49
Optimistic					0	0	109.49
Greedy						0	109.49
RL							0

Table 8.23: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 3, Random delay. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.

	Fast _{departure}	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast _{departure}	0	101.46	101.46	101.46	101.46	101.46	102.32
Conservative		0	100.56	100.56	100.56	100.56	109.73
Cautious			0	0	0	0	109.73
Stable				0	0	0	109.73
Optimistic					0	0	109.73
Greedy						0	109.73
RL							0

Table 8.24: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 3, Unreliable service provider. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.

	Fast _{departure}	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast _{departure}	0	102.41	102.41	102.41	102.41	102.41	101.38
Conservative		0	100.72	100.72	100.72	100.72	109.11
Cautious			0	0	0	0	109.12
Stable				0	0	0	109.12
Optimistic					0	0	109.12
Greedy						0	109.12
RL							0

Table 8.25: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting4, On-time delivery. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.

	Fast _{departure}	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast _{departure}	0	100.67	100.99	101.79	103.03	104.66	105.66
Conservative		0	100.78	101.68	103.00	104.77	105.86
Cautious			0	101.35	102.94	105.28	106.89
Stable				0	102.61	105.54	107.77
Optimistic					0	103.90	106.24
Greedy						0	103.22
RL							0

Table 8.26: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting4, Random delay. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.

	Fast _{departure}	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast _{departure}	0	100.97	101.10	101.62	102.44	103.51	103.91
Conservative		0	101.01	101.51	102.42	103.61	104.29
Cautious			0	101.36	102.50	104.22	105.66
Stable				0	102.40	104.55	107.00
Optimistic					0	103.48	106.32
Greedy						0	104.48
RL							0

Table 8.27: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting4, Unreliable service provider. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.

	Fast _{departure}	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast _{departure}	0	100.63	101.00	101.64	102.88	104.33	105.55
Conservative		0	100.79	101.46	102.82	104.42	105.90
Cautious			0	101.29	102.75	104.94	107.09
Stable				0	102.51	105.15	107.97
Optimistic					0	103.69	106.70
Greedy						0	104.20
RL							0

Table 8.28: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting5, On-time delivery. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.

	Fast _{departure}	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast _{departure}	0	102.16	106.53	108.67	109.44	109.56	110.87
Conservative		0	104.56	106.79	107.66	107.90	109.03
Cautious			0	102.74	103.74	104.02	104.78
Stable				0	101.62	101.97	102.80
Optimistic					0	100.93	102.26
Greedy						0	102.58
RL							0

Table 8.29: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting5, Random delay. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.

	Fast _{departure}	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast _{departure}	0	102.21	105.74	107.37	107.81	107.92	108.82
Conservative		0	104.13	105.81	106.38	106.54	107.33
Cautious			0	102.60	103.27	103.44	104.17
Stable				0	101.66	101.89	103.02
Optimistic					0	101.15	102.86
Greedy						0	103.35
RL							0

Table 8.30: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting5, Unreliable service provider. Each cell [i][j] represents the ratio of the mean performance value from 5000 simulation runs following baseline i to that of baseline j.

	Fast _{departure}	Conservative	Cautious	Stable	Optimistic	Greedy	RL
Fast _{departure}	0	102.08	106.08	108.16	108.86	108.98	110.07
Conservative		0	104.26	106.44	107.23	107.48	108.39
Cautious			0	102.73	103.65	103.91	104.61
Stable				0	101.57	101.87	102.97
Optimistic					0	100.93	102.63
Greedy						0	103.07
RL							0

8.3 Full Strategy Comparison for Settings 4 and 5

Table 8.31: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 4, On-time delivery.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL	Generative
Fast_departure	0	101.00	102.42	103.80	103.84	103.42	102.41	101.62
Conservative		0	101.98	103.38	103.47	103.17	102.33	101.46
Cautious			0	102.09	102.46	102.50	101.70	101.17
Stable				0	101.40	101.78	101.30	101.04
Optimistic					0	101.14	100.98	101.09
Greedy						0	100.90	101.13
RL							0	101.22
Generative								0

Table 8.32: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 4, Random delay.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL	Generative
Fast_departure	0	101.43	102.85	104.37	104.51	104.01	102.34	102.03
Conservative		0	102.42	103.96	104.11	103.78	102.23	102.01
Cautious			0	102.53	102.97	102.98	102.00	101.64
Stable				0	101.79	102.27	101.95	101.38
Optimistic					0	101.49	101.93	101.31
Greedy						0	102.15	101.72
RL							0	101.77
Generative								0

Table 8.33: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 4, Unreliable service provider.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL	Generative
Fast_departure	0	101.04	102.56	104.13	104.31	104.19	103.31	102.38
Conservative		0	102.09	103.67	103.93	103.93	103.18	102.34
Cautious			0	102.25	102.82	103.18	102.66	101.89
Stable				0	101.69	102.46	102.27	101.79
Optimistic					0	101.53	101.86	101.67
Greedy						0	101.89	101.97
RL							0	101.22
Generative								0

Table 8.34: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 5, On-time delivery.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL	Generative
Fast_departure	0	100.44	100.97	101.43	102.05	103.95	102.83	103.50
Conservative		0	100.95	101.47	102.28	104.01	102.99	103.55
Cautious			0	101.22	101.98	103.38	102.57	103.40
Stable				0	101.40	102.65	102.07	103.06
Optimistic					0	101.97	101.57	102.37
Greedy						0	100.93	101.44
RL							0	101.37
Generative								0

Table 8.35: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 5, Random delay.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL	Generative
Fast_departure	0	100.76	101.04	101.50	102.23	103.95	103.16	103.14
Conservative		0	101.10	101.67	102.46	104.21	103.19	103.41
Cautious			0	101.50	102.27	103.71	102.78	103.06
Stable				0	101.76	103.03	102.51	102.69
Optimistic					0	102.36	102.21	102.14
Greedy						0	101.83	101.20
RL							0	101.18
Generative								0

Table 8.36: Table of pairwise relative performance of the different baselines for 5000 simulation runs, Setting 5, Unreliable service provider.

	Fast_departure	Conservative	Cautious	Stable	Optimistic	Greedy	RL	Generative
Fast_departure	0	100.51	101.64	102.63	103.41	104.96	103.31	103.81
Conservative		0	101.38	102.27	103.23	104.80	103.31	103.80
Cautious			0	101.52	102.47	103.90	102.84	103.55
Stable				0	101.59	102.98	102.35	103.09
Optimistic					0	102.10	101.87	102.58
Greedy						0	101.22	101.62
RL							0	101.13
Generative								0

8.4 Glossary of Hyperparameters

Table 8.37: Agent Behavioral Hyperparameters

Parameter	Strategic Implication
SenseConfig: Controls Information-Gathering Strategy	
boards_per_centipawn	The exchange rate between two sensing goals: reducing board set uncertainty versus improving the score of the next move. A higher value prioritizes clarification of the board state.
expected_outcome_coef	Optimism factor. A higher value makes the agent focus more on the average-case outcome when choosing a sense square.
worst_outcome_coef	Pessimism factor. A higher value makes the agent prioritize avoiding the worst possible outcome of a sense action, leading to more cautious sensing.
outcome_variance_coef	Risk aversion. Since this value is typically negative, it penalizes sense actions that lead to a wide, unpredictable range of outcomes, favoring more stable and predictable results.
score_variance_coef	A secondary risk aversion parameter related to the variance in move scores resulting from a sense action.
MoveConfig: Controls Move Selection and Playing Style	
mean_score_factor	Determines the influence of a move’s average score (across all hypothesis boards) on the final decision. Promotes solid, reliable play.

Table 8.37 – continued from previous page

Parameter	Strategic Implication
min_score_factor	Cautiousness factor. A higher value prioritizes moves that are safe even in the worst-case scenario, leading to a defensive style.
max_score_factor	Aggressiveness/Gambling factor. A higher value encourages moves that have a high potential reward, even if they are risky and the best-case scenario is unlikely.
threshold_score	Adds variety to move selection by allowing the agent to choose from any move whose score is within this margin of the best-scoring move. Prevents predictability.
sampling_exploration_coef	Influences the UCT (Upper Confidence bounds applied to Trees) algorithm during move evaluation. A higher value encourages the agent to evaluate moves it has considered less often, preventing premature convergence on a suboptimal choice.
TimeConfig: Controls Time Management	
turns_to_plan_for	Sets the time-planning horizon. A lower value leads to more time spent on early moves, while a higher value conserves time for the endgame.
min_time_for_turn	A safety net to ensure the agent always spends a minimum amount of time thinking, preventing trivial blunders.
max_time_for_turn	A safety net to prevent the agent from spending too much time on a single move and risking a loss on the clock.
time_for_sense	The fraction of a turn's allocated time that should be dedicated to choosing a sense action. A higher value emphasizes the importance of information gathering.

Table 8.37 – continued from previous page

Parameter	Strategic Implication
time_for_move	The fraction of a turn's allocated time dedicated to choosing a move after sensing is complete.

8.5 Details of the Cart-Pole Swing-Up Environment

The Cart-Pole Swing-Up Environment, sourced from the `realworldrl_suite` library and built upon the `dm_control` physics engine. The components of the environment’s underlying Markov Decision Process (MDP) are defined as follows[ope21]:

- **State Space:** The base state is represented by a five-dimensional, continuous vector ($s \in \mathbb{R}^5$). This vector provides a physical description of the system, comprising the cart’s position and velocity, as well as the pole’s angular position (represented by its sine and cosine values) and its angular velocity. It is important to note that this state can be augmented with additional information depending on the configuration, such as a boolean vector indicating the status of safety constraints or noisy features to increase dimensionality.
- **Action Space:** The action space is continuous, consisting of a single scalar value ($a \in \mathbb{R}$) bounded within the normalized range of $[-1, 1]$. This value corresponds to the magnitude and direction of the horizontal force applied to the cart.
- **Reward Function:** A dense and continuous reward is provided at each timestep. The reward value is a function of the pole’s vertical alignment, scaled to be between 0 and 1. A reward of 1 is granted when the pole is perfectly upright, while a reward of 0 is given when the pole is hanging vertically downwards.
- **Episode Termination:** An episode is concluded after a fixed duration. The default time limit is 10 seconds, which, given the simulation’s control timestep, typically corresponds to 1000 discrete steps.

Incorporated Real-World Complexities

A key feature of the `realworldrl_suite` is the ability to introduce challenges that model real-world phenomena. The primary complexities relevant to the configured task are detailed below.

- **Physical Perturbations:** The physical parameters of the environment are non-stationary. The simulation can perturb the pole’s length, the pole’s mass, the joint damping, or the slider’s damping. Under the ‘easy’ challenge setting, the **pole’s length** is specifically perturbed at the start of an

episode, with its value being drawn from a distribution where the length is between **0.9** and **1.1** times its standard length.

- **Safety and Action Constraints:** The environment actively monitors for the violation of predefined constraints. For the swing-up task, these include:
 - **Slider Position:** The cart’s position must remain within a specified range, typically **[-2m, 2m]** from the center of the track.
 - **Balance Velocity:** The pole’s angular velocity must be below a certain threshold when it is near its balanced, upright position.
 - **Slider Acceleration:** The cart’s acceleration is constrained to prevent excessively jarring movements.
 - **Action Rate of Change:** The difference in applied force between consecutive actions is limited to ensure smoother control signals.

8.6 Strategies Employed by Tournament Bots

Below are short descriptions of the bots that participated in the NeurIPS 2022 tournament, which are mentioned in [GPS⁺22]:

StrangeFish2 This bot is a further development of the one with the same name from NeurIPS 2021. All possible locations of opponent pieces are tracked, and for each one, position strengths and likelihoods are estimated. Both sense and move decisions are made to maximize the expected outcome of the turn. Performance was improved through developments in board state value heuristics and by using prioritized sub-sampling of tracked positions when all hypotheses could not be evaluated due to time limits.

Fianchetto Minor changes were made to last year's version of Fianchetto. The bot is based on StrangeFish and uses Leela Chess Zero (Lc0) to model a probability distribution over opponent board states as a POMDP. The most significant change is an increased weight given to defensive sensing and moving. This is done to detect opponent attacking moves that had been assigned a low probability. These weights are specific to the opponent, with more defensive play prioritized against more aggressive opponents. Additionally, its first move is chosen from an opening book designed to favor closed chess positions, based on the idea that such positions have lower uncertainty.

Kevin A distribution over possible game histories is explicitly tracked, as opponent senses are considered important, not only the board state. Opponent actions for both senses and moves are predicted using a neural network. This network was trained with imitation learning on game logs from StrangeFish2. Inputs from OpenSpiel are passed through a residual neural network (ResNet) and a long short-term memory (LSTM) network, which then outputs a distribution over legal actions. The best move is selected by maximizing the expected value over the distribution of histories, and board states are evaluated using Stockfish.

Châteaux This bot is based on deep synoptic Monte Carlo planning. Information states are approximated using a lossy stochastic abstraction, where sets of boards are encoded into compact, fixed-size synopses. These fixed-size representations are used as input to a residual neural network that outputs a value estimation and a policy distribution. The network was trained on approximately 600,000 games. A "second-order" belief state of the opponent's information is also maintained using an unweighted particle filter composed of possible infor-

mation states from the opponent’s view.

ROOKie A comprehensive multi-hypothesis-tracking of piece positions is performed, and actions are selected by voting among the tracked boards. For each tracked board, votes are contributed for up to 5 moves and against any moves that leave the king in check. Computing the top moves this way is often much faster than analyzing position strength. Sense actions are chosen to maximize the expected consensus in votes from the boards that remain.

Oracle Board states are exhaustively tracked. A sense action is chosen to identify possible checks, or if none exist, to minimize the expected number of possible board states. The move that is recommended most frequently by Stockfish across all possible board states is chosen.

Marmot A Monte Carlo counterfactual regret minimization (MC-CFR) algorithm is used for sensing and moving. It includes novel modifications that make assumptions about the past, allowing searches to start from mid-game at the cost of soundness. A heuristic evaluation function, which combines Stockfish’s board-evaluation with a tracked uncertainty measure, is used to evaluate intermediate states. All possible opponent board states for current and past timesteps are tracked and used to start its MC-CFR searches.

JKU-CODA Contextual preference ranking is used to compute a probability distribution over possible board states based on historical games. All possible moves on the most likely boards are evaluated with Stockfish. The move with the best combination of worst-case and weighted-average evaluation is chosen. Sensing is done to maximize the reduction in the number of possible states and to decrease uncertainty in the evaluations of the best move. The ranking is performed by a Siamese neural network that generates an embedding of boards and observations.

DynamicEntropy This bot is a pessimistic version of single observer information set Monte Carlo tree search. The game is considered only from the searching player’s perspective, which allows the simulated opponent to select observations directly instead of its own actions. Leaf nodes are scored by a shallow Stockfish evaluation of randomly sampled board states.

GarrisonNRL A set of possible boards is maintained. The sensing action is selected by finding the location that maximizes a sensing weight, which is the product of the number of potential opponent moves and the value of the pieces moving there. When the number of possible boards is below 50, chess engines

and heuristic methods are used to select a move. For each board, a best move is chosen by attacking the king if possible, or otherwise by using Lc0 with the Maia 1700 network. For larger sets of boards, a single observer information set Monte Carlo tree search approach is employed.

trout (baseline) A single board-state estimate is maintained, which is formed from the latest observation of each square. The move recommended by Stockfish for this board estimate is chosen. If a piece was just captured or a capture is anticipated, sensing is done over the capture square. Otherwise, a random location that does not contain one of its own pieces is selected for sensing.

attacker (baseline) One of four scripted attack sequences is randomly chosen and executed. If that sequence fails, no more moves are made.

random (baseline) Moves are chosen uniformly at random. Sense actions are not relevant.

8.7 Comparative Methods

Below are short descriptions of the second and third methods that participated in the International StepUP Competition for Biometric Footstep Recognition, which are mentioned in [LMP⁺25]:

TI-MFBO framework

To address the high computational expense associated with hyperparameter optimization (HPO) for deep spatio-temporal networks, a framework named Transfer-Initialized Multi-Fidelity Bayesian Optimization (TI-MFBO) is proposed. The sample efficiency of standard Bayesian optimization is improved by this method through the combination of two main ideas: using transfer learning to initialize the surrogate model and applying a multi-fidelity evaluation system. A primary difficulty in Bayesian optimization is the "cold start" problem at the beginning, where the surrogate model has no initial data about the objective function. To overcome this issue, the surrogate model is pre-trained using data from less costly proxy tasks. Examples of these tasks include optimizing a basic 2D-CNN on maximum intensity projections of gait data or using input resolutions that have been significantly down-sampled (*sample_sz*). This use of transfer learning establishes a well-informed prior, which effectively gives the optimization process a "warm start" for the main model.

The TI-MFBO framework operates using an advanced surrogate model, typically a Gaussian Process (GP), for which the prior distribution is informed by the posterior knowledge gained from the proxy task. This method ensures that the first steps in exploring the target hyperparameter space are directed by previous, relevant information. A multi-fidelity structure is used for the optimization, allowing for the flexible allocation of computational resources. Evaluations at low fidelity—such as training on a smaller group of participants or for fewer epochs (*n_epochs*)—are performed to quickly eliminate poor hyperparameter settings. This process is managed by a cost-aware acquisition function, for example, Multi-Fidelity Expected Improvement, which considers the cost of evaluation at every level of fidelity. A system for promotion, similar to successive halving, is used. It guarantees that only the hyperparameter settings that show good results at lower-cost fidelities are advanced for testing at higher-cost fidelities.

By combining an informed starting point with a resource-conscious evaluation method, TI-MFBO greatly cuts down on the number of costly training cycles

needed. This allows for a more comprehensive and efficient search of the hyperparameter space. As a result, the framework is particularly suitable for complex deep learning models that require significant computational resources.

ECCO In conventional HPO, the training process is typically viewed as a fixed black box. An alternative approach, called Evolutionary Curricular Co-Optimization (ECCO), is introduced. This paradigm concurrently optimizes the model's hyperparameters along with the curriculum for the training data. The method is based on the idea that the best hyperparameters might depend on the specific strategy used for presenting data to the model. An evolutionary algorithm is utilized by ECCO, in which every individual within the population stands for a full learning strategy. The genome of each individual contains two different sets of parameters: (1) the model's hyperparameters, such as learning rate, optimizer, and layer sizes (*lr*, *optimizer*, *layer_sizes*), and (2) the curriculum schedule. The curriculum is structured as a series of stages where the data becomes increasingly complex. For example, an early stage might use data from simple walking conditions only (e.g., barefoot (BF) at one speed (W1)), with later stages gradually adding more difficult variables like different footwear (ST, P1, P2) and speeds (W2, W3, W4).

For practical application, the curriculum part of the genome is structured as a list of tuples, $[(epoch_1, conditions_1), (epoch_2, conditions_2), \dots]$, which specifies when the dataset should be expanded. The evolutionary process makes use of special genetic operators; for instance, simulated binary crossover is used for continuous hyperparameters such as learning rate, while uniform crossover is applied for categorical selections like the optimizer. Mutations might include small changes to numerical values or adjustments to the epoch boundaries within the curriculum. An individual's fitness is assessed using a two-part objective that balances the final performance on the validation set against the computational cost, thus favoring strategies that are both effective and efficient. Over multiple generations involving selection, crossover, and mutation, ECCO develops cooperative combinations of curricula and hyperparameters. This process can find effective and efficient training paths that are not discoverable by methods that optimize these elements separately.