Université de Lille
École Graduée MADIS-631, Mathématiques-Sciences du numérique et de leurs interactions

# Thèse de Doctorat

Spécialité **Informatique**

présentée par
**Yann Berthelot**

---

# Efficient Deep Reinforcement Learning for Industrial Process Control

---

## Apprentissage par renforcement profond efficace pour le contrôle de procédés industriels

---

sous la direction de **Philippe Preux**
et d'**Antoine Canaguier-Durand**,
ainsi que l'encadrement de **Riad Akrour**.

---

Soutenue publiquement à **Villeneuve d'Ascq**, le **19 juin 2026** devant le jury composé de

| | | |
|---|---|---|
| M^me^ Ann **Nowé** | Professeure, Vrije Universiteit Brussel | Présidente du jury |
| M. Bruno **Scherrer** | Chargé de recherche HDR, Inria | Rapporteur |
| M. Sylvain **Lamprier** | Professeur des universités, Université d'Angers | Rapporteur |
| M. Adam **White** | Associate Professor, University of Alberta | Examinateur |
| M. Philippe **Preux** | Professeur des universités, Université de Lille | Directeur de thèse |
| M. Antoine **Canaguier-Durand** | Chef de projet R&D, Saint-Gobain Research Paris | Co-Directeur de thèse |
| M. Riad **Akrour** | Chargé de recherche HDR, Inria | Encadrant de thèse |

---

Centre de Recherche en Informatique, Signal et Automatique de Lille (CRIStAL),
UMR 9189 Équipe Scool, 59650, Villeneuve d'Ascq, France

SAINT-GOBAIN Université de Lille CRIStAL Centre de Recherche en Informatique, Signal et Automatique de Lille Inria

# Remerciements

Je pense que tout travail scientifique est autant le fruit du travail de ses auteurs que de celui de nos prédécesseurs et des collaborations nouées en chemin. La thèse n'y fait donc pas exception. J'irai même plus loin, c'est sûrement le moment où, construisant nos compétences en recherche et notre autonomie, nous sommes le plus portés par le soutien de notre entourage. Ainsi, j'aimerais remercier dans ces quelques lignes les personnes sans qui les autres milliers de lignes de cette thèse n'auraient pu voir le jour.

Tout d'abord, une thèse est autant un sujet de recherche que le cadre qui l'entoure. La mienne a eu cela de particulier qu'étant une thèse CIFRE, elle reposait sur la collaboration entre deux structures réunies au sein d'une seule et même direction de thèse. Je tiens donc à remercier chaleureusement mes directeurs de thèse, Philippe et Antoine, ainsi que mon encadrant, Riad. Au-delà de l'évident soutien que vous m'avez apporté sur les volets scientifiques et techniques, je veux aussi souligner le soutien humain et moral, la compassion et l'écoute, dont vous avez fait preuve durant plus de trois années. Un remerciement tout particulier à Philippe et Antoine, à qui s'est ajoutée la charge du montage de ce projet de recherche. Au-delà du soutien que vous m'avez apporté durant ces trois ans, je vous reste infiniment reconnaissant de m'avoir fait confiance pour ce projet en premier lieu.

Derrière la direction de thèse se trouve naturellement le reste des deux équipes avec lesquelles j'ai eu la chance de collaborer. Mes remerciements vont ensuite à l'équipe Scool, autant pour les aspects scientifiques que pour la bonne humeur et la camaraderie qui y ont toujours été au rendez-vous. Je veux tout particulièrement remercier Timothée, pour sa patience et sa pédagogie sur les questions de statistiques, de reproductibilité, et Adrienne pour ses nombreux cours improvisés sur divers concepts mathématiques. Je tiens aussi à remercier Brahim, Waris, Julien et Hector pour les nombreuses discussions (souvent même productives) que nous avons pu avoir (et les nombreux tableaux remplis et effacés qui les accompagnent) et l'aide qu'ils ont pu m'apporter. Je remercie également l'équipe Coda de Saint-Gobain Research Paris, et tout particulièrement deux personnes sans qui cette thèse n'aurait pas vu le jour : Gwen pour m'avoir soutenu et avoir cru en moi dès ma candidature, et Patrick pour avoir été un soutien clé de mon parcours vers la recherche et m'avoir guidé vers Saint-Gobain et cette thèse.

I am deeply grateful to the members of the jury for agreeing to review my work and for their valuable comments and advice, and particularly to Sylvain Lamprier and Bruno Scherrer for reporting on this dissertation.

Enfin, je tiens bien évidemment à remercier mes parents et mon frère pour leur soutien indéfectible et leur présence à mes côtés durant (et même avant !) ces trois années.

# Résumé

Le contrôle de procédés industriels, tel que la régulation de température dans un four à verre de type « float », présente des défis caractéristiques : une physique complexe, une observabilité partielle et un coût d'opération élevé. Cette thèse étudie comment rendre l'apprentissage par renforcement (RL) profond plus robuste, plus efficace en termes d'échantillons et mieux adapté aux environnements industriels réels, en tant qu'alternative viable aux contrôleurs classiques (PID, MPC).

Nous commençons par formaliser le contrôle de température du four comme une tâche de RL caractérisée par six propriétés structurelles, dont l'observabilité partielle, le délai d'actionnement et la disponibilité de contrôleurs sous-optimaux existants. L'expérimentation sur le système réel étant impossible, nous identifions la simulation du contrôle d'altitude d'un avion en 2D comme environnement de substitution partageant toutes ces propriétés. Nous le publions au sein de TargetGym, une bibliothèque open-source de benchmarks de suivi de consigne, écrite en JAX.

Un premier défi est la non-stationnarité de la politique au cours de l'entraînement, qui dégrade la plasticité du réseau de valeur. Nous introduisons EVarEst, une fonction de perte qui pénalise la variance des erreurs résiduelles, préservant la plasticité et améliorant les performances de SAC de 13,29 % en moyenne par rapport à l'objectif MSE standard sur les benchmarks MuJoCo et AdroitHand.

Un second défi est de combler l'écart entre la simulation et la réalité. L'entraînement sur des environnements massivement parallèles permet d'induire une robustesse à la mauvaise spécification du modèle comme propriété émergente. Ces travaux sont soutenus par ProbingEnvironments, une suite de tests d'intégration pour les algorithmes acteur-critique, et AJAX, une bibliothèque de RL pour l'exécution GPU de bout en bout atteignant jusqu'à 68× la vitesse des frameworks séquentiels standards.

Enfin, nous formalisons le suivi de consigne sous la forme de MDPs de Consigne (SP-MDPs) et prouvons que les politiques biais-optimales sont nécessairement stables, faisant des contrôleurs classiques existants des guides d'exploration naturels. Sur cette base, nous proposons EDGE (*Expert Decayed Guided Exploration*), qui mélange stochastiquement les actions de l'expert et de la politique selon un programme décroissant, agissant comme une surcouche légère sur tout algorithme hors-politique. EDGE améliore les performances asymptotiques de 17,0 % par rapport à SAC et converge jusqu'à 26,5 % plus rapidement, surpassant une variante d'apprentissage par renforcement résiduel de SAC (+11,1 %) et une baseline de type IBRL argmax (+12,7 %) en performance asymptotique, tout en restant robuste aux imperfections de l'expert.

# Abstract

Industrial process control, such as temperature regulation in a float glass furnace, presents problems that are physically complex, partially observable, and costly to interact with. This thesis investigates how to make deep reinforcement learning (RL) more robust, more sample-efficient, and better suited to real industrial settings, as a viable alternative to classical controllers (PID, MPC).

We begin by formalizing furnace temperature control as an RL task characterized by six structural properties, including partial observability, actuation delay, and the availability of existing suboptimal controllers. Because direct experimentation on the real system is infeasible, we identify 2D aircraft altitude control as a simulation proxy sharing all these properties, which we release as part of TargetGym, an open-source library of JAX-native setpoint-tracking benchmarks.

A first challenge is policy non-stationarity during training, which degrades the value network's plasticity. We introduce EVarEst, a value loss that penalizes the variance of residual errors, preserving plasticity and improving SAC performance by 13.29% on average over the standard MSE objective on MuJoCo and AdroitHand benchmarks.

A second challenge is bridging the simulation-to-real gap. Training across massively parallel environments induces robustness to model misspecification as an emergent property. ProbingEnvironments, an integration test suite for actor-critic algorithms, and AJAX, an RL library for end-to-end GPU execution achieving up to 68$\times$ speedup over standard sequential frameworks, support this work.

Finally, we formalize setpoint-tracking as Setpoint MDPs (SP-MDPs) and prove that bias-optimal policies are necessarily stable, making existing classical controllers natural exploration guides. Building on this, we propose EDGE (Expert Decayed Guided Exploration), which stochastically mixes expert and policy actions under a decaying schedule, acting as a lightweight wrapper on any off-policy algorithm. It achieves a 17.0% improvement in asymptotic performance over SAC and converges up to 26.5% faster, outperforming a Residual RL variant of SAC (+11.1%) and an IBRL-style argmax baseline (+12.7%) in asymptotic performance, while remaining robust to expert imperfections.

# Contents

**List of Acronyms** xi

**1 Introduction** 1

1.1 Outline and Contributions . . . 3

**2 Background and Notations** 7

2.1 The Reinforcement Learning Framework . . . 8

2.2 Deep Reinforcement Learning . . . 12

2.3 Algorithms Used in This Thesis . . . 17

2.4 Summary . . . 22

**I Case Study: Glass Furnace** 25

**3 Problem Statement** 27

3.1 The Float Glass Process . . . 28

3.2 Furnace Control and Its Current Limitations . . . 33

3.3 Why Reinforcement Learning? . . . 34

3.4 Modeling the Furnace as a Control Problem . . . 34

3.5 Interaction or Simulation . . . 38

3.6 Identification of a Proxy Problem . . . 38

3.7 Implementation . . . 41

3.8 Conclusion . . . 41

**4 Literature Review** **43**

4.1 Reinforcement Learning for Industrial Process Control . . . . . 44

4.2 Sample-Efficient Reinforcement Learning . . . . . 49

4.3 Robust Value Estimation in Deep RL . . . . . 56

4.4 Conclusion . . . . . 58

**II Robust Reinforcement Learning** **61**

**5 Robust Reinforcement Learning** **63**

5.1 Introduction . . . . . 65

5.2 Related works . . . . . 66

5.3 Background and notations . . . . . 67

5.4 Error-Variance penalized Estimation (EVarEst) . . . . . 69

5.5 Experimental study . . . . . 72

5.6 Conclusion . . . . . 80

**III Efficient Reinforcement Learning** **83**

**6 Parallelization in Reinforcement Learning** **85**

6.1 Algorithmic parallelization . . . . . 86

6.2 Implementing parallelization . . . . . 92

6.3 Conclusion . . . . . 103

**7 Reliable and Efficient Implementations: Building the Tools for This Thesis** **105**

7.1 Reliability in RL . . . . . 106

7.2 AJAX . . . . . 113

7.3 TargetGym . . . . . 119

7.4 Conclusion . . . . . 125

**IV Reinforcement Learning for Setpoint tracking** **127**

**8 Expert-guided Deep Reinforcement Learning for Setpoint Tracking** **129**

8.1 Introduction . . . 131
8.2 Related Work . . . 132
8.3 Formalization . . . 134
8.4 Method . . . 137
8.5 Experiments . . . 142
8.6 Conclusion . . . 153

**9 General Conclusion and Perspectives** **157**

9.1 Context and Objectives . . . 157
9.2 Contributions . . . 157
9.3 Research Questions Revisited . . . 160
9.4 Limitations . . . 162
9.5 Perspectives . . . 163

**A Complements on Chapter 5** **167**

A.1 Proof of lemmas . . . 167
A.2 Additional figures . . . 168
A.3 Critic comparison protocol . . . 173
A.4 Hyperparameters . . . 173

**B Longitudinal Flight Dynamics Model** **177**

B.1 Introduction . . . 177
B.2 Coordinate System and State Variables . . . 177
B.3 Atmospheric Model . . . 179
B.4 Aerodynamic Forces . . . 180
B.5 Lifting Surfaces and Pitch Moment Balance . . . 182
B.6 Thrust Model . . . 183

B.7 Equations of Motion . . . 184
B.8 Actuator Dynamics . . . 185
B.9 Numerical Integration . . . 186
B.10 Physical Parameters Summary . . . 186
B.11 Summary of Modeling Approximations . . . 186
B.12 Observation, Action, and Reward . . . 188

**C Complements on Chapter 8** **191**

C.1 Proofs . . . 191
C.2 Stable Policy Identification in Stable-SP-MDPs . . . 193
C.3 Baselines pseudocode . . . 194
C.4 Effect of TD-target bootstrap on Imitation Bootstrapped Reinforcement Learning (IBRL) and EDGE . . . 199
C.5 Statistical test methodology . . . 201
C.6 Hardware and Computations . . . 204

**D Use of Generative AI Tools** **207**

**List of Figures** **210**

**List of Algorithms** **216**

**List of Tables** **217**

**List of References** **219**

# List of Acronyms

**A**

**A2C** Advantage Actor-Critic , 19, 48, 88

**A3C** Asynchronous Advantage Actor-Critic , 48, 87, 88

**ACER** Actor-Critic with Experience Replay , 19, 210

**AJAX** Agents in JAX: Python Reinforcement Learning Library in JAX , 105, 112–121, 125, 126, 143, 159, 161, 164, 213

**API** Application Programming Interface , 111, 112

**APO** Average Policy Optimization , 114

**AR-RL** Average-Reward Reinforcement Learning

**ASAC** Average-reward Soft Actor-Critic , 114, 118

**AVEC** Actor with Variance Estimated Critic , 63, 65–67, 70, 71, 74–77, 80, 158, 161, 217

**AVG** Action Value Gradient , 114, 117

**B**

**BPTT** Backpropagation Through Time , 55

**Brax** A Differentiable Physics Engine for Large Scale Rigid Body Simulation , 117

**C**

**CEM** Cross-Entropy Method , 54, 55

**CI** Confidence Interval , 144, 146, 151, 200, 213–215

**CI** Continuous Integration , 118

**CleanRL** High-quality single-file implementations of Deep Reinforcement Learning algorithms , 114

**CPU** Central Processing Unit , 94, 113, 115, 117, 120, 121, 217

**CrossQ** CrossQ: Batch Normalization in Deep Reinforcement Learning for Greater Sample Efficiency , 51

**CSTR** Continuous Stirred-Tank Reactor , 121–124, 213

**CUDA** Compute Unified Device Architecture , 95

**D**

**DAgger** Dataset Aggregation , 133, 141

**DDPG** Deep Deterministic Policy Gradient , 19, 48, 106, 112, 113

**DMC** DeepMind Control Suite , 54

**DQN** Deep Q-Network , 17, 19, 48, 56, 67, 85, 87

**DroQ** Dropout Q-Learning , 51

**E**

**EDGE** Expert Decayed-Guided-Exploration , 5, 159, 160, 162–164, 209

**EVarEst** Error-Variance penalized Estimation viii, 63–67, 69–81, 113, 158, 161–164, 168–174, 209, 211, 212, 214, 215, 217

**F**

**FQI** Fitted Q-Iteration , 13, 56, 65

**FWER** Family-Wise Error Rate , 203

**G**

**GAE** Generalized Advantage Estimation , 15, 20, 21, 72, 118

**GAIL** Generative Adversarial Imitation Learning , 133

**GPU** Graphics Processing Unit , 3, 5, 85, 88, 89, 94, 95, 105, 113–115, 117, 118, 120, 121, 125, 126, 159, 161, 204, 212

**Gymnasium** An API standard for reinforcement learning with a diverse collection of reference environments , 94, 212

**H**

**HLO** High-Level Operations (XLA intermediate representation) , 96

**I**

**i.i.d.** independent and identically distributed , 57

**IBRL** Imitation Bootstrapped Reinforcement Learning x, 133, 140, 143, 145, 147, 149, 151–153, 159, 160, 162, 163, 199, 200, 215

**IL** Imitation Learning , 132, 133, 138

**IMPALA** Importance Weighted Actor-Learner Architecture , 88

**IQM** Interquartile Mean , 144–146, 151, 153, 155, 160, 162, 200, 202, 213–215

**J**

**JAX** JAX: Composable transformations of Python+NumPy programs , 3, 105, 114–121, 197, 213

**JIT** Just In Time Compilation , 114, 117, 120, 121, 197, 217

**L**

**LLVM** Low-Level Virtual Machine , 96

**LQR** Linear Quadratic Regulator , 44, 47

**LSTM** Long Short-Term Memory , 45

**LTL** Linear Temporal Logic , 132

**M**

**MBPO** Model-Based Policy Optimization , 54

**MBRL** Model-Based Reinforcement Learning , 52

**MCTS** Monte Carlo Tree Search , 53, 54

**MDP** Markov Decision Process , 4, 5, 8–10, 56, 109, 130, 132, 134–137, 191, 193

**ML** Machine Learning , 106

**MLP** Multi-Layer Perceptron , 197

**MPC** Model Predictive Control , 2, 33, 44, 45, 47, 90, 129, 131, 132, 136, 153, 154, 157, 160

**MPO** Maximum A-Posteriori Policy Optimization , 45, 89

**MPPI** Model Predictive Path Integral , 90

**MSE** Mean Squared Error , 3, 4, 63, 65–67, 69–71, 73, 75–81, 158, 161, 170, 212, 214

**MuJoCo** Multi-Joint dynamics with Contact physics simulator , 54, 63, 66, 73, 94, 114, 115, 154, 158, 161, 212

**MushroomRL** Python Reinforcement Learning Library , 114

**N**

**NEA** Normalized Expert Advantage , 136, 144–146, 151, 152, 155, 160, 162, 200, 204, 213–215, 217, 218

**NMPC** Nonlinear Model Predictive Control , 47

**P**

**PID** Proportional–Integral–Derivative controller , 2, 33, 44, 47, 90, 129, 131, 134, 136, 151, 153, 154, 157, 160

**POMDP** Partially Observable Markov Decision Process , 9

**PPO** Proximal Policy Optimization , 3–5, 14, 15, 18–22, 48, 56, 57, 63, 66, 72–76, 78, 79, 85, 87, 89, 106, 107, 112, 114–117, 158, 161, 170, 211–214, 216, 217

**PQL** Parallel Q-Learning , 89

**PQN** Parallelized Q-Network , 89

**PTX** Parallel Thread Execution , 96

**R**

**Ray Rllib** Python Reinforcement Learning Library , 114

**REDQ** Randomized Ensembled Double Q-learning , 16, 17, 19, 20, 51, 114, 118

**RL** Reinforcement Learning , 1–5, 7, 8, 10, 12–14, 17, 19, 22, 23, 63, 65–72, 81, 85, 94, 105, 106, 109, 111, 112, 114, 118, 119, 125, 126, 129, 131, 132, 134, 135, 138, 140, 153–155, 157–165, 210, 212

**RRL** Residual Reinforcement Learning , 132, 134, 138, 143, 145, 151–154, 159, 160

**RSSM** Recurrent State Space Model , 55

**S**

**SAC** Soft Actor-Critic , 3–5, 14, 15, 17–22, 51, 54, 56, 63, 66, 72–76, 78–80, 89, 91, 92, 102, 112, 114–117, 132, 139, 142, 143, 145–155, 158, 160–163, 209–214, 216, 217

**SB3** Stable Baselines3: Python Reinforcement Learning Library , 72, 94, 111–117, 119–121, 159, 212, 213

**SGD** Stochastic Gradient Descent , 65

**sim2real** Simulation-to-Real , 151

**SIMD** Single Instruction, Multiple Data , 95

**SPT** Setpoint Tracking , 11, 129–132, 153–155, 159, 160, 163, 164

**SQL** Soft Q-Learning , 19, 210

**SVP** Sample Variance Penalization , 67, 71

**T**

**TD** Temporal Difference , 13, 133

**TD3** Twin Delayed Deep Deterministic Policy Gradient , 16–19, 89, 133, 143

**TPE** Tree-structured Parzen Estimator , 196

**TPU** Tensor Processing Unit , 120

**TRPO** Trust Region Policy Optimization , 18–20, 106, 107, 112

**U**

**UTD** Update-To-Data ratio , 16, 20, 50, 51, 55, 114

**V**

**VRE** Variance of Residual Errors , 63, 65–67, 69–71, 76, 78, 80, 81, 158, 161, 170, 212, 214

**W**

**W&B** Weights & Biases: a machine learning experiment tracking platform , 118

**X**

**XLA** Accelerated Linear Algebra , 96, 103

# Chapter 1

# Introduction

> « En essayant continuellement, on finit par réussir… Donc plus ça rate, plus on a de chances que ça marche. »
> *"By constantly trying, one eventually succeeds... So the more it fails, the better the chances that it will work."*
>
> Les Shadoks — Rouxel (1968–1974)

This opening quotation from "Les Shadoks" involuntarily but perfectly describes the underlying process behind Reinforcement Learning (RL): learning by trial and error. In the absurd world of the Shadoks, this translates into failing as fast as possible[1]. In the world of RL, however, the quotation rings just as true: we pride ourselves on being more efficient than brute force, yet the more we try, the more we learn and the better we get. Indeed, the industrious and industrial nature of the Shadoks finds an echo in both the method and the subject of this thesis, as it is the product of 3 years of work devoted to the improvement of industrial process control using RL.

Industrial process control encompasses a broad class of engineering problems in which a physical system must be driven toward and maintained at a desired operating point. Such problems pervade modern manufacturing: chemical reactors, cement kilns, paper mills, and glass furnaces all require precise, continuous regulation of temperature, pressure, or flow rate. The scale at which these processes operate makes control accuracy a matter of global consequence: industry accounts for 24% of direct global greenhouse gas emissions, rising to 34% once indirect emissions from electricity and heat production are included (Dhakal et al., 2022). Inaccurate control manifests in two ways: at best, unnecessary overshoots that waste energy directly; at worst, quality defects that trigger product rejection and reprocessing, compounding both costs and emissions. This thesis is conducted in collaboration with Saint-Gobain[2], one

[1] If something has a one-in-a-million chance of succeeding, the Shadoks rush through the 999,999 failures to reach the successful millionth attempt, see Season 1 Episode 27 (Rouxel, 1968)

[2] More specifically, Saint-Gobain Research Paris

of the world's largest glass manufacturers, producing approximately 6 million tons of glass per year across furnace lines hundreds of meters long, at temperatures exceeding 1500°C. The specific application is temperature control inside a float glass furnace.

Temperature control in such furnaces is currently managed through manual operator decisions, Proportional–Integral–Derivative (PID, Ziegler and Nichols (1942)) controllers, or Model Predictive Control (MPC, García, Prett, and Morari (1989)). These classical approaches achieve satisfactory performance but are fundamentally limited: PID controllers involve a small number of parameters that are difficult to tune and cannot easily adapt to changing operating conditions, while MPC depends on an accurate forward model of the system, a model that is itself expensive to derive and maintain. RL offers a compelling alternative: it can learn a control policy directly from interaction, without requiring an explicit model, and can optimize over far more complex behavior spaces than classical controllers. Until recently, however, successful RL applications to real physical systems remained largely confined to research settings. A landmark result came in 2022, when Degrave et al. demonstrated real-time RL control of a tokamak plasma (a system of comparable complexity, partial observability, and safety criticality) using 5,000 parallel actors (Degrave et al., 2022). This result, alongside the rapid maturation of deep RL tooling, constitutes a strong signal that RL is becoming a viable candidate for complex industrial control.

Despite this momentum, a structural gap persists between the RL research community and industrial control practice. Practitioners with deep domain knowledge but limited RL expertise produce application-oriented work that brings little algorithmic novelty. Conversely, RL researchers focused on standard benchmarks (locomotion, game-playing) have limited exposure to the defining properties of industrial control: setpoint tracking objectives, partial observability, long actuation delays, non-stationarity, and the existence of functioning but suboptimal classical controllers. The absence of accessible benchmarks for this problem class is itself a barrier: without standardized environments, the two communities have no common ground, and improvements remain isolated rather than cumulative. This thesis sits at the intersection of both worlds, studying how RL methods must be adapted to be effective in this setting and providing the infrastructure that makes such study accessible.

A fundamental constraint of this work is that the glass furnace cannot be used for direct experimentation: it operates continuously at industrial scale, and the cost and safety implications of exploratory RL training make direct interaction impossible. Identifying a simulation proxy that faithfully captures all the structural properties of the furnace, while remaining accessible to the broader RL community, is therefore not a minor problem but a **central contribution of this thesis**. We propose 2D aircraft altitude control as such a proxy: a physical system that shares the furnace's setpoint tracking objective, partial observability, actuation delay, non-stationarity, and stable equilibrium structure, while being cheap to simulate and requiring no glass-making

expertise. This proxy forms the experimental backbone of the thesis and is released as part of TargetGym, an open-source suite of hardware-accelerated environments for setpoint-tracking control research.

These observations led us to consider five research questions. (1) What structural properties characterize industrial process control as an RL problem, and how can they be used to rigorously justify the choice of a simulation proxy? (2) How can value network training be made robust to the policy non-stationarity inherent to online RL? (3) How can parallelism make deep RL training tractable for industrial tasks, which require far more experience than standard benchmarks and whose less-studied nature demands extensive experimentation, and what further properties does it induce? (4) What software infrastructure is needed for correct and reproducible RL research on industrial benchmarks? (5) How can existing classical controllers be integrated into RL training to improve both sample efficiency and final performance on setpoint tracking tasks?

The following section presents the contributions that address each question in turn.

## 1.1 Outline and Contributions

The contributions of this thesis are:

- **A formal characterization and simulation proxy for industrial process control.** We identify six structural properties that define the furnace control problem as an RL task, and propose 2D aircraft altitude control as a proxy that shares all of them, making systematic algorithmic research on this problem class possible without access to the real system (Chapter 3).
- **EVarEst**, a generalized value loss that reweights the bias-variance decomposition of the standard Mean Squared Error (MSE) objective, improving value network plasticity under policy non-stationarity and yielding average performance gains of 13% with Soft Actor-Critic (SAC, Haarnoja, A. Zhou, Abbeel, et al. (2018)) and 28% with Proximal Policy Optimization (PPO, Schulman, Wolski, et al. (2017)) across MuJoCo and AdroitHand benchmarks (Chapter 5).
- **Three open-source software tools** for reliable and efficient RL research (Chapter 7): **ProbingEnvironments**, an integration test suite for actor-critic algorithms; **AJAX**, a JIT compilation Autograd XLA (JAX)-based RL library for end-to-end Graphics Processing Unit (GPU) and seed-parallel training; and **TargetGym**, a suite of JAX-native setpoint-tracking environments including the aircraft altitude control proxy used throughout this thesis.

- **A formalization of setpoint tracking as a class of Markov Decision Processes (MDPs).** We identify the structural properties that distinguish setpoint tracking from standard RL benchmarks, prove that stable policies (those whose trajectories ultimately reach and remain within the target region) are gain-optimal and that all bias-optimal policies must be stable, and use this result to derive principled expert integration (Chapter 8).
- **EDGE** (Expert Decayed-Guided-Exploration), an algorithm that integrates a partial expert controller into the RL exploration process via stochastic action mixing, improving both sample efficiency and asymptotic performance on setpoint tracking while remaining robust to imperfections in the expert (Chapter 8).

This thesis is organized around three concerns reflected in its title: applying RL to industrial process control, making that application robust, and making it efficient. The following chapters address each concern in turn.

**Chapter 2: Background and Notations.** We introduce the RL formalism used throughout the thesis. We cover the key distinctions that structure the field: how the source of training data (on-policy vs. off-policy) constrains algorithm design, why learning a world model is appealing but often impractical for complex systems (model-free vs. model-based), and how the choice of policy gradient estimator determines what information the critic must provide. We then present PPO and SAC, the two algorithms central to this thesis, and situate them within a taxonomy that makes their structural differences explicit.

**Chapter 3: Problem Statement.** We introduce the industrial application that motivates this thesis: temperature control inside a float glass furnace. We characterize its six defining properties as a control problem, argue why each one makes a direct RL deployment on the real system impractical, and provide the structural justification for using 2D aircraft altitude control as the experimental proxy throughout the thesis.

**Chapter 4: Literature Review.** We survey the three fields that converge in this thesis. We first review existing applications of RL to industrial process control, drawing lessons about what has worked and what remains unsolved. We then review sample-efficient RL methods, distinguishing model-free approaches that increase the update-to-data ratio from model-based approaches that leverage a learned world model, and arguing that model-free approaches are better suited to our setting. Finally, we review the literature on robust value estimation in deep RL, documenting the consequences of policy non-stationarity on value networks and identifying the gap that Chapter 5 addresses.

**Chapter 5: Robust Reinforcement Learning.** We study the impact of policy non-stationarity on value network training and propose EVarEst, a new critic objective that reweights the bias and variance terms of the standard MSE decomposition. EVarEst subsumes existing objectives as special cases and improves value network adaptability to successive policy updates. We

provide empirical evidence on the MuJoCo and AdroitHand benchmarks that it improves over both SAC and PPO baselines.

**Chapter 6: Parallelization in Reinforcement Learning.** We examine how parallelism, both at the algorithmic level and at the implementation level, can improve the time efficiency of deep RL. We discuss the structure of parallel data collection and the computational bottlenecks that limit scaling. We further survey the history of parallel RL algorithms and show that massive parallelization yields robustness properties beyond mere speed, and discuss how this facilitates transfer from simulation to real industrial systems.

**Chapter 7: Reliable and Efficient Implementations.** We argue that the gap between algorithm pseudocode and its implementation is a primary and underappreciated source of irreproducibility in deep RL, and that closing it is a prerequisite for trustworthy research on industrial benchmarks. We present three open-source tools built to this end: a test suite for catching implementation bugs before full-scale training, a GPU-native training library for fast and reproducible experimentation, and a benchmark suite for setpoint-tracking tasks.

**Chapter 8: Expert-guided Deep Reinforcement Learning for Setpoint Tracking.** We formalize setpoint tracking as a class of target MDPs and establish theoretical properties that motivate expert integration. We then propose Expert Decayed-Guided-Exploration (EDGE), validate it on the aircraft altitude control proxy, and characterize its robustness to imperfections in the expert, a property critical for any real-world deployment where the expert controller is itself imperfect.

**Chapter 9: Conclusion.** We summarize the contributions of this thesis, discuss their limitations, and outline directions for future work, including the path toward deployment on a real float glass furnace.

# Chapter 2

# Background and Notations

> Scientific developments can always be made logical and rational with sufficient hindsight. It is amazing, however, how clouded the crystal ball looks beforehand. We all wear such intellectual blinders and make such inexplicable blunders that it is amazing that any progress is made at all
>
> Eye of the Hurricane: An Autobiography — R. Bellman (1984)

The basic concepts of RL are remarkably intuitive: an agent acts, observes consequences, and learns to do better. The difficulty lies elsewhere. Methods that appear similar at first glance turn out to rest on fundamentally different mechanisms; conversely, concepts with an apparently clear distinction start to merge together, revealing continuums without clear boundaries. The closer one looks, the more clouded the picture becomes, until enough understanding accumulates for the dots to connect and the structure to become clear once more. This is precisely the dynamic Bellman describes above. This chapter introduces the formalism and notation used throughout the thesis, covering the Markov decision process framework, value functions, policy gradient estimators, and the algorithmic families that subsequent chapters build upon. This chapter aims to be accessible to readers new to the field while remaining precise enough to serve as a reference.

## Contents


**2.1 The Reinforcement Learning Framework** . . . . . 8
**2.2 Deep Reinforcement Learning** . . . . . 12
**2.3 Algorithms Used in This Thesis** . . . . . 17
**2.4 Summary** . . . . . 22

## 2.1 The Reinforcement Learning Framework

### 2.1.1 Markov Decision Processes

RL addresses the problem of finding an optimal policy for a system whose dynamics are modeled as an **MDP** (R. Bellman, 1957; Puterman, 1994):

**Definition 2.1** Markov Decision Process

A MDP is a tuple $(\mathcal{S}, \mathcal{A}, P, R)$, where:

- $\mathcal{S}$ is the *state space*, with $s \in \mathcal{S}$ a state;
- $\mathcal{A}$ is the *action space*, with $a \in \mathcal{A}$ an action;
- $P(s' \mid s, a) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the *transition function*, giving the probability of transitioning to $s'$ from $s$ after taking action $a$;
- $R(s, a, s') : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the *reward function*.

An MDP formalizes the interaction between an agent and a dynamical system: it characterizes how states evolve under actions and what rewards are produced, but specifies no objective. It becomes a well-defined optimization problem only once an objective criterion is added, as done in Section 2.1.2. Different criteria (discounted return, average reward, finite horizon) yield different problems from the same MDP.

MDPs were introduced by R. Bellman (1957) over *finite* state and action spaces; Puterman (1994) provides the standard reference for their rigorous formalization. Throughout this thesis, $\mathcal{S} \subseteq \mathbb{R}^{d_s}$ and $\mathcal{A} \subseteq \mathbb{R}^{d_a}$ are continuous, with $d_s$ and $d_a$ the respective dimensions; the time index $t$ remains discrete. This continuous-state/action extension preserves the MDP structure defined above and is the relevant setting for the control problems studied here.

The MDP framework rests on the **Markov property**: the next state depends only on the current state and action, not on the full history (the future, conditioned on the present, is independent of the past),

$$P(s_{t+1} \mid s_0, a_0, \ldots, s_t, a_t) = P(s_{t+1} \mid s_t, a_t).$$

This property makes the problem tractable by ensuring that the current state is a sufficient statistic of the history.

At each time step $t \in \mathbb{N}$, the agent observes state $s_t$, selects action $a_t$, transitions to $s_{t+1} \sim P(\cdot \mid s_t, a_t)$, and receives reward $r_{t+1} = R(s_t, a_t, s_{t+1})$. The sequence $\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \ldots)$ is called a **trajectory**; $\tau \sim \pi$ denotes a trajectory generated by policy $\pi$ (formally defined in Section 2.1.2).

#### Partially Observable Markov Decision Processes

In many real-world settings, the agent does not observe the true state $s_t$, due to hidden state dimensions (lack of sensors) and/or noise. The Partially Observable Markov Decision Process (MDP) framework extends the MDP to account for this partial information structure.

**Definition 2.2** Partially Observable Markov Decision Process

A POMDP extends the MDP $(\mathcal{S}, \mathcal{A}, P, R)$ with two additional components:

- $\mathcal{O} \subseteq \mathbb{R}^{d_o}$ is the *observation space*, with $o \in \mathcal{O}$ an observation and $d_o$ the observation dimension;
- $\Omega(o \mid s, a) : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \to [0, 1]$ is the *observation function*, giving the probability of observing $o$ upon arriving in state $s$ via action $a$.

The agent receives $o_t \sim \Omega(\cdot \mid s_t, a_{t-1})$ in place of $s_t$ at each step $t \geq 1$; the initial observation $o_0$ is drawn from $\Omega(\cdot \mid s_0)$, conditioning only on the initial state since no prior action exists. The underlying state still satisfies the Markov property, but the agent must act on partial information.

This setting is particularly relevant to the industrial applications studied in Chapter 3, where sensor noise and limited instrumentation prevent full state observability.

### 2.1.2 Policies, Returns, and the Optimization Objective

A **policy** is the agent's decision rule: it maps the current state to an action or a distribution over actions.

**Definition 2.3** Policy

A *stochastic policy* $\pi_\theta : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ maps states to probability distributions over actions, parameterized by $\theta \in \mathbb{R}^{d_\theta}$. Action sampling is written $a_t \sim \pi_\theta(\cdot \mid s_t)$.
A *deterministic policy* $\mu_\theta : \mathcal{S} \to \mathcal{A}$ maps each state directly to a single action.

A stochastic policy addresses the exploration/exploitation tension naturally by maintaining a distribution over actions, allowing the agent to sometimes deviate from its current best estimate. A deterministic policy commits to a single action per state and requires separate, explicit exploration mechanisms.

**Exploration and Exploitation** The agent faces a fundamental tension between two incompatible goals: **exploitation** (selecting actions currently believed to yield high reward) and **exploration** (selecting actions whose outcomes are uncertain, in order to gather information that may reveal better alternatives). A purely exploitative agent risks converging to suboptimal

action selection; a purely exploratory one forgoes performance by selecting actions already known to be inferior, treating training as if reward itself did not matter. This tension has no universal solution and is a central design consideration in any RL algorithm.

Because we are dealing with **sequential** decision making, the agent's objective is not to maximize immediate reward, but rather to accumulate reward over time. Formally, let $\rho_0$ denote the initial state distribution ($s_0 \sim \rho_0$). A **reinforcement learning** (RL) problem consists in solving an MDP, paired with an optimization objective, in which neither $P$ nor $R$ is known to the agent; both must be inferred from interaction with the environment. The goal of RL is to find the **optimal policy** (or policies) $\pi^*$ that maximizes some scalar objective $J(\pi)$:

$$\pi^* = \arg\max_{\pi} J(\pi).$$

The choice of objective depends on the problem setting. The most common formulation uses the **discounted return**, which quantifies total future rewards from a given time step, weighting nearer rewards more heavily than distant ones.

**Definition 2.4** Discounted Return

The *discounted return* from time step $t$ is:

$$G_t \triangleq \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}.$$

The discount factor $\gamma$ controls the trade-off between immediate and future rewards: $\gamma \to 0$ renders the agent myopic, while $\gamma \to 1$ weights all future rewards equally.

Discounting ($\gamma < 1$) ensures convergence of the infinite sum in continuing tasks, and reflects the agent's temporal preferences. The corresponding RL objective is then the **expected discounted return**:

$$J(\pi) \triangleq \mathbb{E}_{\substack{s_0 \sim \rho_0 \\ \tau \sim \pi}}[G_0] = \mathbb{E}_{\substack{s_0 \sim \rho_0 \\ \tau \sim \pi}}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1}\right]. \tag{2.1}$$

Alternatives exist for continuing tasks where discounting is not appropriate (Puterman, 1994).

**Definition 2.5** Average-Reward Objective

The *average-reward* objective measures the long-run mean reward per time step:

$$J_{avg}(\pi) \triangleq \lim_{T \to \infty} \frac{1}{T} \mathbb{E}_{\tau \sim \pi}\left[\sum_{t=0}^{T-1} r_{t+1}\right].$$

This criterion treats all time steps equally, avoiding the myopic bias introduced by discounting, and is the natural objective for tasks that run indefinitely without a preferred start state. It is used in Chapter 8 as the basis for the Setpoint Tracking (SPT)-MDP formalization of setpoint tracking problems.

### 2.1.3 Value Functions

**Definition 2.6** State-Value, Action-Value, and Advantage Functions

The *state-value function* $V^\pi : \mathcal{S} \to \mathbb{R}$ gives the expected return from state $s$ under policy $\pi$:

$$V^\pi(s) \triangleq \mathbb{E}_{\tau\sim\pi}[G_t \mid s_t = s]\,.$$

The *action-value function* $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ gives the expected return after taking action $a$ in state $s$ and then following $\pi$:

$$Q^\pi(s,a) \triangleq \mathbb{E}_{\tau\sim\pi}[G_t \mid s_t = s,\ a_t = a]\,.$$

The *advantage function* $A^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ measures how much better action $a$ is compared to what the policy $\pi$ would achieve on average from state $s$ (i.e., $V^\pi(s)$):

$$A^\pi(s,a) \triangleq Q^\pi(s,a) - V^\pi(s).$$

Intuitively, $V^\pi(s)$ measures how much total reward the agent expects to collect from state $s$ onwards under policy $\pi$. $Q^\pi(s,a)$ extends this to also condition on the first action, allowing comparison across actions in a given state. $A^\pi(s,a)$ measures how much better action $a$ is compared to what the policy $\pi$ would achieve on average from state $s$ (i.e., $V^\pi(s)$), providing a centered signal for policy improvement.

These functions satisfy the **Bellman expectation equations**:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_{a\sim\pi,\, s'\sim P}\left[R(s,a,s') + \gamma V^\pi(s')\right], \\ Q^\pi(s,a) &= \mathbb{E}_{s'\sim P}\left[R(s,a,s') + \gamma\, \mathbb{E}_{a'\sim\pi}\left[Q^\pi(s',a')\right]\right]. \end{aligned}$$

The optimal value functions $V^*(s) = \max_\pi V^\pi(s)$ and $Q^*(s,a) = \max_\pi Q^\pi(s,a)$ satisfy the **Bellman optimality equations** (Sutton and A. Barto, 2020), from which the optimal policy is recovered as $\pi^*(a \mid s) = \arg\max_a Q^*(s,a)$. This maximization is tractable only in discrete action spaces of reasonable size; in large discrete spaces or continuous spaces it must be approximated, as discussed in Section 2.3.1.

## 2.2 Deep Reinforcement Learning

The quantities defined above, $\pi$, $V^\pi$, and $Q^\pi$, are functions over continuous spaces $\mathcal{S}$ and $\mathcal{A}$. *Tabular* methods represent them as lookup tables, which is only feasible when both spaces are finite and small. In the continuous, high-dimensional settings considered in this thesis, tabular representations are intractable: the state space alone may be $\mathbb{R}^{d_s}$ with $d_s$ in the tens or hundreds.

**Deep reinforcement learning** addresses this by approximating these functions with neural networks[1]:

- the policy is parameterized as $\pi_\theta(a \mid s)$ or $\mu_\theta(s)$, with $\theta$ the network weights; the neural network implementing the policy is called the **actor**;
- the value functions are parameterized as $V_\phi(s)$ and $Q_\phi(s, a)$, with $\phi$ the network weights; the neural network implementing the value function is called the **critic**.

Neural networks generalize across states through learned feature representations, making RL applicable to large and continuous spaces. Neural networks introduce approximation error, but this is preferable to the alternative of discretizing the spaces, which suffers from the curse of dimensionality. All algorithms used in this thesis adopt this neural network parameterization; the subscripts $\theta$ and $\phi$ refer to actor and critic network weights respectively throughout.

In practice, computing $V^\pi$ or $Q^\pi$ analytically is intractable, so $V_\phi$ and $Q_\phi$ must be estimated from data. The two principal objectives used to train such estimates are:

**Definition 2.7** Monte Carlo Return Error

The *Monte Carlo return error* trains $V_\phi$ to match the observed discounted return:

$$\mathcal{L}_{MC}(\phi) = \mathbb{E}_{(s_t, G_t) \sim \tau}\left[(V_\phi(s_t) - G_t)^2\right].$$

The target $G_t$ is unbiased but high-variance, since it depends on the entire subsequent trajectory.

[1] While the "Deep" in Deep Reinforcement Learning refers to deep learning and deep neural networks (meaning neural networks with a large number of hidden layers), in practice the networks used in many Deep RL algorithms are relatively shallow, usually 2 layers. Nonetheless, the name "Deep" is the de-facto term for the use of neural function approximation in RL.

**Definition 2.8** Temporal-Difference (TD) Error

The *Temporal Difference* (*TD*) *error* (Sutton, 1988) measures the one-step inconsistency in the Bellman equation. For a transition $(s, a, r, s')$ and a target action $a^-$ produced by a target policy $\pi^-$:

$$\delta \triangleq r + \gamma\, Q_\phi(s', a^-) - Q_\phi(s, a).$$

The choice of $\pi^-$ varies by algorithm: on-policy methods (SARSA) sample $a^- \sim \pi(\cdot \mid s')$; Q-learning methods (DQN) set $a^- = \arg\max_{a'} Q_\phi(s', a')$; actor-critic methods (TD3, SAC) compute $a^-$ from the actor network. Minimizing the expected squared TD error yields the value-estimation objective:

$$\mathcal{L}_{TD}(\phi) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(Q_\phi(s, a) - \underbrace{\left(r + \gamma\, Q_\phi(s', a^-)\right)}_{\text{TD target}}\right)^2\right].$$

Bootstrapping introduces bias but substantially reduces variance compared to Monte Carlo estimation.

Minimizing $\mathcal{L}_{TD}(\phi)$ is unstable: the TD target depends on the same network being updated, so the target shifts with every gradient step. The **target network** (Volodymyr Mnih, Kavukcuoglu, D. Silver, Graves, et al., 2013), a periodically updated copy $Q_{\bar{\phi}}$ of $Q_\phi$, addresses this by replacing $Q_\phi(s', a^-)$ in the TD target with $Q_{\bar{\phi}}(s', a^-)$, freezing it for several gradient steps. This reduces the correlation between inputs and targets and makes the updates closer to those of a stationary supervised regression.

#### Critic Training in Practice

In practice, value networks are trained under the Fitted Q-Iteration (FQI, Ernst, Geurts, and Wehenkel (2005)) scheme, which frames value estimation as a sequence of supervised regression problems. A mini-batch of transitions $(s, a, r, s')$ is sampled from past experience, the TD target $r + \gamma Q_{\bar{\phi}}(s', a^-)$ is computed using the frozen target network $Q_{\bar{\phi}}$, and one gradient step is taken to minimize $\mathcal{L}_{TD}(\phi)$. The target network parameters $\bar{\phi}$ are then updated toward $\phi$, either by periodic hard copy or by an exponential moving average. The practical challenges that arise from this training scheme, in particular the non-stationarity of the regression target and overestimation bias, are the subject of Chapter 5.

### 2.2.1 On-Policy and Off-Policy Learning

A fundamental distinction between RL algorithms stems from the source of the data that can be used for their updates.

**Definition 2.9** Replay Buffer

A *replay buffer* $\mathcal{D} = \{(s_i, a_i, r_i, s'_i)\}_{i=1}^{|\mathcal{D}|}$ is a finite-capacity dataset of past transitions collected during training. At each update step, a mini-batch is sampled uniformly from $\mathcal{D}$, decoupling data collection from learning and enabling the reuse of each transition across multiple gradient updates.

**Definition 2.10** On-Policy and Off-Policy

An algorithm is **on-policy** if updates are restricted to transitions collected under the current policy $\pi_\theta$. Transitions generated by earlier policy versions must be discarded after each update; using stale data would introduce a distribution mismatch between the data and the current policy, violating the on-policy assumption and potentially destabilizing training. An algorithm is **off-policy** if it is capable of using transitions collected under any behavior policy, including transitions stored in a replay buffer accumulated over the course of training.

On-policy algorithms (e.g., PPO) have a more straightforward theoretical analysis in tabular and linear function approximation settings (Sutton and A. Barto, 2020) and lower implementation complexity, but discard data after each policy update. Off-policy algorithms (e.g., SAC) reuse past transitions many times, substantially improving sample efficiency at the cost of added complexity: they require maintaining a replay buffer, managing the interplay between behavior and target policies, and carefully tuning the update-to-data ratio.

This distinction is relevant to all contributions of this thesis, as discussed in Chapters 5 to 8.

### 2.2.2 Policy Gradient Estimation

The **policy gradient** $\nabla_\theta J(\pi_\theta)$ is the gradient of the RL objective with respect to the policy parameters. All policy-gradient algorithms optimize the policy by ascending an estimate of this gradient. Two fundamental families of estimators exist, differing in how this gradient is computed:

**Definition 2.11** Likelihood-Ratio Estimator

The *likelihood-ratio* (or score function) estimator expresses the policy gradient as:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot \Psi_t\right],$$

where $\Psi_t$ is a scalar return estimate. Common choices are the discounted return $G_t$ (R. J. Williams, 1992) or the advantage estimate $\hat{A}_t^\pi \approx A^\pi(s_t, a_t)$ when a critic is available (Schulman, Moritz, et al., 2018). The estimator requires only $\nabla_\theta \log \pi_\theta$; it does not require gradients through the value function with respect to actions.

**Definition 2.12** Pathwise (Reparameterization) Estimator

The *pathwise* estimator reparameterizes the action as a deterministic function of the state and a noise variable: $a = f_\theta(s, \varepsilon), \varepsilon \sim p(\varepsilon)$. This yields (T. P. Lillicrap et al., 2016; Haarnoja, A. Zhou, Abbeel, et al., 2018):

$$\nabla_\theta J(\pi_\theta) \approx \mathbb{E}_{s,\varepsilon}\Big[\nabla_a Q^\pi(s,a)\big|_{a=f_\theta(s,\varepsilon)} \cdot \nabla_\theta f_\theta(s,\varepsilon)\Big].$$

Gradients flow back through the Q-function with respect to actions, which requires $Q^\pi$ to be differentiable in $a$, and therefore restricts this estimator to **continuous** action spaces.[a] This estimator typically exhibits lower variance than the likelihood-ratio estimator (Haarnoja, A. Zhou, Abbeel, et al., 2018), at the cost of relying on an accurate Q-function.

[a] The Gumbel-Softmax trick (Jang, Gu, and Poole, 2017; Maddison, A. Mnih, and Teh, 2017) provides a differentiable relaxation of discrete sampling, allowing a form of reparameterization in discrete spaces. However, it introduces a temperature-controlled approximation bias and remains less standard in practice than the likelihood-ratio estimator for discrete action spaces.

**Definition 2.13** Generalized Advantage Estimation

*Generalized Advantage Estimation* (*GAE*) (Schulman, Moritz, et al., 2018) provides a practical family of advantage estimators parameterized by $\lambda \in [0, 1]$. Define the one-step TD residual:

$$\delta_t \triangleq r_{t+1} + \gamma V_\phi(s_{t+1}) - V_\phi(s_t).$$

The GAE estimate is the exponentially weighted sum of these residuals:

$$\hat{A}_t^{\mathrm{GAE}(\lambda)} \triangleq \sum_{\ell=0}^{\infty} (\gamma\lambda)^\ell \, \delta_{t+\ell}.$$

For $\lambda = 0$, this reduces to the one-step TD residual $\delta_t$ (high bias, low variance); for $\lambda = 1$, it becomes the full Monte Carlo return minus a baseline (low bias, high variance). Intermediate values of $\lambda$ interpolate between these extremes, trading off bias and variance.

PPO (Schulman, Wolski, et al., 2017) uses the likelihood-ratio estimator with $\Psi_t = \hat{A}_t^\pi$ computed via GAE. SAC uses the pathwise estimator, reparameterizing the action through a squashed

Gaussian policy (Haarnoja, A. Zhou, Abbeel, et al., 2018). Twin Delayed Deep Deterministic Policy Gradient (TD3, Fujimoto, Hoof, and Meger (2018)) uses a deterministic variant of the pathwise estimator, $f_\theta(s, \varepsilon) = \mu_\theta(s)$, so the gradient reduces to $\nabla_\theta \mu_\theta(s)^\top \nabla_a Q^\pi(s, \mu_\theta(s))$ (T. P. Lillicrap et al., 2016).

### 2.2.3 Sample Efficiency

**Sample efficiency** refers to the amount of environment interaction required for an agent to reach a given level of performance (Sutton and A. Barto, 2020). A sample-efficient algorithm achieves high performance with fewer transitions. This matters whenever interactions are expensive, whether due to the cost of physical experiments, the time required to run high-fidelity simulators, or safety constraints that limit exploratory behavior. Improving sample efficiency is a central concern of this thesis: the industrial settings studied in Chapters 3 to 8 involve either expensive simulators or real plants, where data collection is a primary bottleneck.

Beyond the on-policy/off-policy distinction, the **Update-To-Data (UTD, X. Chen et al. (2021))** ratio, the number of gradient updates performed per real[2] environment transition, is a key lever for sample efficiency. High-UTD off-policy methods such as Randomized Ensembled Double Q-Learning (REDQ) (X. Chen et al., 2021) push sample efficiency further by performing many updates per collected transition. However, increasing the UTD ratio indiscriminately is not free: each additional update consumes more compute per step, and high UTD ratios risk overfitting the critic to the current replay buffer, which can destabilize training and degrade asymptotic performance.

### 2.2.4 Model-Free and Model-Based Reinforcement Learning

A further distinction concerns whether the agent has access to, or learns, a **model of the environment**.

**Definition 2.14** Environment Model

An *environment model* consists of a learned approximation $\hat{P}(s' \mid s, a)$ of the transition function and, optionally, an approximation $\hat{R}(s, a, s')$ of the reward function.

**Model-based** methods use such a model to plan ahead or to generate synthetic transitions (Sutton and A. Barto, 2020, Chapter 8). They can be highly sample-efficient (in terms of real-world transitions), since the learned model may be queried many times per real environment step. However, inaccuracies in the model accumulate over multi-step rollouts, potentially degrading policy quality.

[2] As opposed to simulated environment in model-based methods, see below for model-based methods

**Model-free** methods learn directly from real environment transitions, without explicitly modeling $P$ or $R$. They are generally simpler and by definition robust to model errors, at the cost of usually requiring more environment interactions (see Chapter 4 for model-free methods that recover competitive sample efficiency via high update-to-data ratios). This thesis focuses on model-free methods applied to continuous-control tasks; the difficulty of learning accurate dynamics models for the industrial processes considered motivates this choice, as discussed in Chapter 4.

### 2.2.5 Maximum Entropy Reinforcement Learning

SAC (Haarnoja, A. Zhou, Abbeel, et al., 2018) is one of the most widely adopted algorithms for continuous control and is a central method in this thesis. Its foundation is the **maximum entropy** framework, which we introduce here before presenting the full SAC algorithm in Section 2.3.3.

Maximum entropy RL augments the standard RL objective (Equation (2.1)) with a policy entropy bonus:

$$J_{MaxEnt}(\pi_\theta) \triangleq \mathbb{E}_{\substack{s_0\sim\rho_0 \\ \tau\sim\pi_\theta}}\left[\sum_{t=0}^{\infty}\gamma^t\big(r_{t+1} + \alpha\,\mathcal{H}(\pi_\theta(\cdot \mid s_t))\big)\right],$$

where $\mathcal{H}(\pi_\theta(\cdot \mid s)) = -\mathbb{E}_{a\sim\pi_\theta}[\log \pi_\theta(a \mid s)]$ is the differential entropy of the policy (quantifying how spread-out the action distribution is) and $\alpha > 0$ is a temperature parameter controlling the trade-off between reward and entropy maximization. Encouraging high entropy prevents premature convergence to a deterministic policy and promotes broader exploration. The corresponding Bellman equation incorporates the entropy term at each step, yielding the soft Bellman equation:

$$Q^\pi_{soft}(s,a) \triangleq \mathbb{E}_{s'\sim P}\Big[R(s,a,s') + \gamma\,\mathbb{E}_{a'\sim\pi}\Big[Q^\pi_{soft}(s',a') - \alpha\log\pi(a' \mid s')\Big]\Big]. \tag{2.2}$$

SAC (Haarnoja, A. Zhou, Abbeel, et al., 2018) and REDQ (X. Chen et al., 2021) optimize this objective; this is what distinguishes their "Soft Bellman on $Q$" row in Figure 2.1 from the standard Bellman row used by TD3 and Deep Q-Network (DQN, Volodymyr Mnih, Kavukcuoglu, D. Silver, Graves, et al. (2013)).

## 2.3 Algorithms Used in This Thesis

### 2.3.1 A Taxonomy of Deep Reinforcement Learning Algorithms

Beyond the model-free/model-based distinction, deep RL algorithms differ along three structural dimensions that are worth making explicit before surveying individual methods. The taxonomy presented here organizes these dimensions systematically; making them explicit up-

front avoids the common confusion of treating algorithmic differences as mere hyperparameter choices.

**Role of the critic** In likelihood-ratio methods (e.g., PPO), the critic estimates $V^\pi(s)$ purely as a **baseline** to compare against: rather than weighting actions by raw discounted returns $\hat{G}_t$, computing the advantage $\hat{A}_t^\pi = \hat{G}_t - V^\pi(s_t)$ subtracts a state-dependent baseline, which reduces the variance of the gradient estimator. The critic parameters $\phi$, however, receive no gradient from the actor loss ($\phi$ is treated as fixed when differentiating with respect to $\theta$). Removing the critic recovers REINFORCE, so while it is beneficial for variance reduction, the critic is ultimately auxiliary. Conversely, in pathwise methods (e.g., TD3, SAC), the critic estimates $Q^\pi(s,a)$ and the policy gradient is $\nabla_a Q^\pi$ projected back through the actor: via $\nabla_\theta \mu_\theta$ for a deterministic actor $\mu_\theta(s)$ (as in TD3), or via $\nabla_\theta f_\theta$ for a stochastic actor $f_\theta(s,\varepsilon)$ that produces actions by transforming noise $\varepsilon \sim \mathcal{N}(0,I)$ (as in SAC). Without the critic, no gradient exists.

**Interpretation of the gradient signal** The likelihood-ratio gradient weights each action by the scalar $\hat{A}_t^\pi$, measuring whether that action was better or worse than average under $\pi$. The pathwise gradient, by contrast, provides a directional signal in action space: $\nabla_a Q^\pi(s,a)$ points toward higher-valued actions, and is then projected to parameter space via $\nabla_\theta \mu_\theta$ or $\nabla_\theta f_\theta$ (as defined above). The two signals are structurally different even when they produce the same policy update direction.

**The data regime as a structural property** The likelihood-ratio estimator computes an expectation **under the current policy** $\pi_\theta$. Using transitions from older policies introduces a distribution mismatch that can only be corrected with importance weights, a correction that is stable only for small policy changes (Schulman, Levine, et al., 2015; Schulman, Wolski, et al., 2017), as in Trust Region Policy Optimization (TRPO, Schulman, Levine, et al. (2015)) or PPO's clipped surrogate. Off-policy reuse across a full replay buffer is not realistically feasible without introducing large policy changes. The pathwise estimator, by contrast, evaluates $Q^\pi$ at $(s,a)$ pairs where $a \sim \pi_\theta(\cdot \mid s)$ is sampled from the current policy via reparameterization. The provenance of $s$ does not affect the actor gradient, provided it comes from the replay buffer, i.e., from the current or a past version of the policy: the state may come from any past interaction, making replay buffers structurally compatible. The stored action $a$ from the buffer is used only for the critic update, while the actor update re-samples $\tilde{a} \sim \pi_\theta(\cdot \mid s)$ from the current policy.

Figure 2.1 summarizes these algorithms along the three structural axes. Boxed bold names are algorithms central to this thesis; contextual algorithms are shown without a box.

Policies that derive actions as $\arg\max_a Q^*(s,a)$ are tractable only in discrete action spaces. For continuous control, this maximization must be approximated: a deterministic actor $\mu_\theta(s)$ approximates $\arg\max_a Q^\pi(s,a)$ via gradient ascent on $Q^\pi$, which is precisely the deterministic

| Critic objective | Policy Gradient Estimator | | |
|---|---|---|---|
| | **None** | **Likelihood-ratio** | **Pathwise** |
| **None** | | REINFORCE | |
| **MC/GAE on $V$** | | A2C<br>TRPO<br>**PPO** | |
| **Bellman on $Q$** | DQN | ACER | DDPG<br>**TD3** |
| **Soft Bellman on $Q$** | SQL | | **SAC**<br>**REDQ** |

**Figure 2.1** – Deep RL algorithm map organized by PG estimator (columns) and critic objective (rows). on-policy and off-policy cells encode the data regime; gray cells represent structurally incompatible or purely tabular combinations. **Boxed bold** names are algorithms central to this thesis; plain names provide structural context. Actor-Critic with Experience Replay (ACER) (off-policy likelihood-ratio with Bellman $Q$) remains niche because importance sampling variance dominates at high replay ratios. Soft Q-Learning (SQL) (Soft Q-Learning) uses a soft Bellman objective with an implicit policy derived directly from the $Q$-function rather than a separate actor network; it has been largely superseded by SAC.

pathwise estimator. DQN and TD3 are therefore two realizations of the same idea, namely analytical versus learned argmax, separated by the discrete/continuous divide. Note that the pathwise estimator is applicable only in continuous action spaces, since it requires differentiability of $Q$ with respect to $a$. Likewise, Advantage Actor-Critic (A2C, Yuhuai Wu, Mansimov, Grosse, et al. (2017)), TRPO, and PPO occupy the same structural position; their differences are implementation choices (advantage normalization, trust-region constraint, clipped surrogate) layered on top of the same likelihood-ratio gradient.

Actor-critic algorithms can appear superficially similar when viewed from the outside: they all maintain a policy network updated via some gradient signal and a value network that assists that update. PPO and SAC are a prominent example, and the same apparent similarity applies to most algorithm pairs in Figure 2.1. Their separation along all three structural dimensions

(the gradient estimator, the critic objective, and the data regime) reflects genuine structural differences, not merely hyperparameter choices: different estimator families, different value functions, and different data regimes, each following directly from those foundational choices.

The primary algorithms used in this thesis are PPO and SAC, which represent these two paradigms and are used throughout Chapters 5 to 8. REDQ extends SAC with a high UTD ratio to further improve sample efficiency and is surveyed in Chapter 4.

### 2.3.2 Proximal Policy Optimization

PPO (Schulman, Wolski, et al., 2017) is an on-policy, likelihood-ratio algorithm. At each iteration it collects a batch of transitions using the current policy, computes GAE advantages (Section 2.2.2), and then performs multiple gradient update epochs over that batch before discarding it.[3] The key innovation over its predecessor TRPO is the clipped surrogate objective, which implicitly enforces a trust region without requiring constrained optimization. Letting $r_t(\theta) = \pi_\theta(a_t \mid s_t)/\pi_{\theta_{\text{old}}}(a_t \mid s_t)$ denote the probability ratio (not the reward; the notation follows Schulman, Wolski, et al. (2017)), the actor loss is:

$$\mathcal{L}^{\text{CLIP}}(\theta) = \mathbb{E}_t\Big[\min\Big(r_t(\theta)\,\hat{A}_t,\ \text{clip}(r_t(\theta),\,1{-}\varepsilon,\,1{+}\varepsilon)\,\hat{A}_t\Big)\Big],$$

where $\varepsilon$ is a small hyperparameter (typically $0.2$). Clipping prevents the policy from moving too far from $\pi_{\theta_{\text{old}}}$ in a single update. For $K$ epochs, mini-batches of size $M$ are re-sampled from the $N$ collected transitions before the entire batch is discarded. The critic is trained in parallel to minimize a GAE-bootstrapped value target. Algorithm 1 gives the full procedure.

[3] Strictly speaking, the likelihood-ratio gradient is unbiased only on the first epoch, since subsequent epochs update $\theta$ away from $\theta_{\text{old}}$, introducing a distribution mismatch. The clipped surrogate objective limits how far $\pi_\theta$ can drift from $\pi_{\theta_{\text{old}}}$ within an iteration, implicitly enforcing a KL-like constraint that keeps this mismatch small in practice.

**Algorithm 1** Proximal Policy Optimization (PPO)

**Initialize** policy $\pi_\theta$, critic $V_\phi$
**for** each iteration **do**
  **– Data collection –**
  Collect $N$ transitions $(s_t, a_t, r_{t+1}, s_{t+1})$ using $\pi_\theta$; store $\log \pi_\theta(a_t \mid s_t)$
  Compute GAE advantages $\hat{A}_t$ and value targets $\hat{V}_t \triangleq \hat{A}_t + V_\phi(s_t)$ using $\gamma$, $\lambda$
  $\theta_{\text{old}} \leftarrow \theta$
  **for** $K$ epochs **do**
    **for** each mini-batch $\mathcal{B} = \{(s_i, a_i, r_{i+1}, s_{i+1})\}_{i=1}^{M}$ sampled from the collected transitions **do**
      $r_t(\theta) \leftarrow \pi_\theta(a_t \mid s_t) \;/\; \pi_{\theta_{\text{old}}}(a_t \mid s_t)$
      $\theta \leftarrow \theta + \nabla_\theta \mathbb{E}_{\mathcal{B}}\Big[\min\Big(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1{-}\varepsilon, 1{+}\varepsilon)\hat{A}_t\Big)\Big]$
      $\phi \leftarrow \phi - \nabla_\phi \mathbb{E}_{\mathcal{B}}\Big[(V_\phi(s_t) - \hat{V}_t)^2\Big]$
    **end for**
  **end for**
**end for**

### 2.3.3 Soft Actor-Critic

SAC (Haarnoja, A. Zhou, Abbeel, et al., 2018) is an off-policy, pathwise algorithm that optimizes the maximum entropy objective (Section 2.2.5). It maintains a replay buffer $\mathcal{D}$, two critics $Q_{\phi_1}$ and $Q_{\phi_2}$ with corresponding target networks, and a stochastic actor parameterized as a squashed Gaussian (a Gaussian whose samples are passed through $\tanh$ to bound the actions). Taking the minimum of the two critic targets, a trick introduced by Fujimoto, Hoof, and Meger (2018) to suppress overestimation bias, is augmented here with the entropy bonus from Equation (2.2). The temperature $\alpha$ (introduced in Section 2.2.5) can be fixed or learned automatically by treating the entropy constraint $\mathbb{E}_{(s,a)\sim\pi_\theta}[\log \pi_\theta(a \mid s)] \geq \bar{\mathcal{H}}$ as a Lagrangian problem, where $\bar{\mathcal{H}}$ is a target entropy. A common heuristic (Haarnoja, A. Zhou, Abbeel, et al., 2018) sets $\bar{\mathcal{H}} = -\dim(\mathcal{A})$, i.e., the negative of the number of action dimensions, though this choice has no universal justification and may require tuning in practice. Target networks are updated by an exponential moving average with smoothing rate $\tau \in (0, 1)$ (typically $\tau = 0.995$, so the target network moves slowly toward the current network); note that $\tau$ here is a scalar hyperparameter, distinct from the trajectory variable $\tau \sim \pi$ used in Section 2.1.2. Algorithm 2 gives the full procedure.

**Algorithm 2** Soft Actor-Critic (SAC)

**Initialize** policy $\pi_\theta$, critics $Q_{\phi_1}$, $Q_{\phi_2}$, targets $\bar{Q}_{\phi_1} \leftarrow Q_{\phi_1}$, $\bar{Q}_{\phi_2} \leftarrow Q_{\phi_2}$, buffer $\mathcal{D}$, temperature $\alpha$
**for** each environment step $t$ **do**
  $a_t \leftarrow f_\theta(s_t, \varepsilon_t), \quad \varepsilon_t \sim \mathcal{N}(0, I)$ (reparameterized sampling)
  Execute $a_t$, observe $r_{t+1}$, $s_{t+1}$; store $(s_t, a_t, r_{t+1}, s_{t+1})$ in $\mathcal{D}$
  Sample mini-batch $\{(s_i, a_i, r_i, s'_i)\}_{i=1}^{M} \sim \mathcal{D}$ (all updates below are expectations over this mini-batch)
  **– Critic update –**
  $a' \leftarrow f_\theta(s', \varepsilon'), \quad \varepsilon' \sim \mathcal{N}(0, I)$
  $y \leftarrow r + \gamma\Big[\min_{i=1,2} \bar{Q}_{\phi_i}(s', a') - \alpha \log \pi_\theta(a' \mid s')\Big]$
  $\phi_i \leftarrow \phi_i - \nabla_{\phi_i} \mathbb{E}_{\mathcal{B}}\big[(Q_{\phi_i}(s, a) - y)^2\big]$ for $i = 1, 2$
  **– Actor update –**
  $\tilde{a} \leftarrow f_\theta(s, \varepsilon), \quad \varepsilon \sim \mathcal{N}(0, I)$
  $\theta \leftarrow \theta + \nabla_\theta \mathbb{E}_{\mathcal{B}}[\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}) - \alpha \log \pi_\theta(\tilde{a} \mid s)]$
  **– Temperature update –**
  $\alpha \leftarrow \alpha - \nabla_\alpha \mathbb{E}_{\mathcal{B}}\Big[\alpha\Big(-\log \pi_\theta(\tilde{a} \mid s) - \bar{\mathcal{H}}\Big)\Big]$
  **– Target network update –**
  $\bar{Q}_{\phi_i} \leftarrow \tau\, \bar{Q}_{\phi_i} + (1 - \tau)\, Q_{\phi_i}$ for $i = 1, 2$ ($\tau$: target-network smoothing rate)
**end for**

## 2.4 Summary

This chapter has introduced the formalism that the rest of the thesis builds upon. The MDP framework provides the common language: states, actions, and rewards; the discounted return criterion (Section 2.1.2) transforms this into an optimization problem. Value functions express how much future reward a policy can expect and are central to both the robust critic training studied in Chapter 5 and the expert integration developed in Chapter 8. The on-policy/off-policy distinction determines what data an algorithm may use: off-policy methods improve sample efficiency by reusing past transitions from a replay buffer, while on-policy methods, though more data-hungry, are better suited to massive parallelism on hardware accelerators due to their lighter memory footprint. These two approaches represent complementary paths to making RL practical on real systems, as discussed in Chapter 6. The likelihood-ratio and pathwise gradient estimators, instantiated here as PPO and SAC, sit at opposite ends of this divide and are the two algorithms used throughout all experimental chapters. The model-free choice made explicit here is revisited in Chapter 4, where the difficulty of learning accurate dynamics models for industrial processes is used to argue against model-based alternatives.

With this formalism in place, the next chapter turns to the specific industrial problem that motivates this thesis: temperature control inside a float glass furnace, and the structural properties that make it a challenging and distinctive RL problem.

# Part I

# Case Study:
# Glass Furnace

# Chapter 3

# Problem Statement

> The fact that the polynomial is an approximation does not necessarily detract from its usefulness because all models are approximations. Essentially, all models are wrong, but some are useful
>
> Box and Draper (1987)

As this thesis takes an interest in industrial process control (Seborg et al., 2016), we will begin by giving our definition of this concept. While processes found in the industry vary greatly in nature, a key class of them revolves around driving and/or maintaining an industrial system in a desired state. Controlling the concentration of chemicals inside a reactor, maintaining the temperature in a furnace, controlling the flow rate of a liquid, are all examples of process control. While we are interested in industrial applications, the class of problems actually extends outside of plants. The notion of real-life physics coupled with setpoint tracking is the core component of this problem class.

## Contents


**3.1 The Float Glass Process** . . . . . 28
**3.2 Furnace Control and Its Current Limitations** . . . . . 33
**3.3 Why Reinforcement Learning?** . . . . . 34
**3.4 Modeling the Furnace as a Control Problem** . . . . . 34
**3.5 Interaction or Simulation** . . . . . 38
**3.6 Identification of a Proxy Problem** . . . . . 38
**3.7 Implementation** . . . . . 41
**3.8 Conclusion** . . . . . 41

## 3.1 The Float Glass Process

The example of industrial process used throughout this work is a glass-making process called float (Pilkington, 1959)[1]. This process allows for the creation of flat sheets of glass by pouring molten glass over tin. As tin is denser than molten glass, the molten glass floats on top of the tin (hence the name float), allowing it to spread evenly under its own weight and create a flat sheet with a smooth surface. The float process at Saint-Gobain is of a massive scale, producing around 6 000 000 tons of float glass every year (Saint-Gobain, 2021), and using temperatures reaching between 1550°C and 1600°C, with throughput of around 1000$m^2$ per minute, formed on a 500m long line. This scale implies that every improvement to the efficiency of this process, however small, can lead to massive savings in the end. On the other hand, mistakes are also very costly. The float process is divided into four main parts: raw material preparation, the furnace, the lehr, and the cold end.

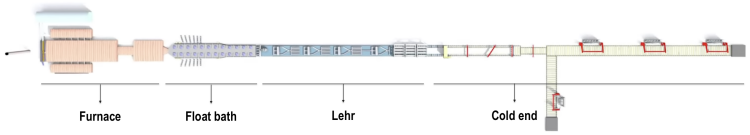



**Figure 3.1** – Top down view of the whole flat glass production line. From Saint-Gobain, 2021

### 3.1.1 Raw Material Preparation

Raw material preparation is the process during which the materials that will constitute the glass (primarily sand, soda, lime, with addition of dolomite and sulfate) are mixed to the required proportions for the current batch. To this basic mix, additives are also added to reach the desired properties for the glass (mechanical, thermal or optic properties, as well as color). In addition, recycled glass and cullet (crushed or broken glass) are added to the raw ingredients. The cullet primarily originates from the scrap cuts of the float process itself, thereby reducing waste. Prior to being fed into the downstream furnace, these materials are stored in silos. The different recipes for different kinds of glass, the use of recycled glass and cullet in varying proportions, and the impact of the storage of the raw materials, are multiple sources of variability in the properties of the input materials. The most impactful property for our furnace control task is the variation in humidity of the material, as it will modify the quantity of energy needed to melt it, by first requiring energy to evaporate the water. The mixture, called batch, is then fed into the furnace; the flow rate at which it is fed is one of the controlled properties of the process.

[1] A video presentation of this process can be seen at https://www.youtube.com/watch?v=1HDWJgFLCfA


(a) Raw materials

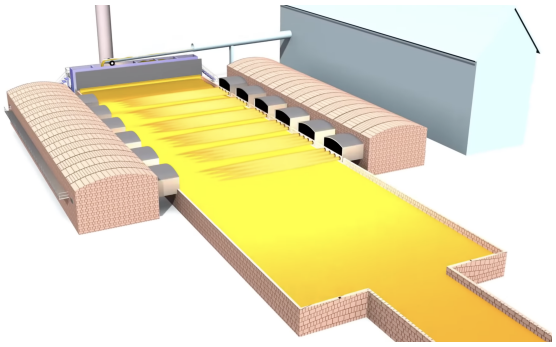

(b) Cullet


(c) Batch charger

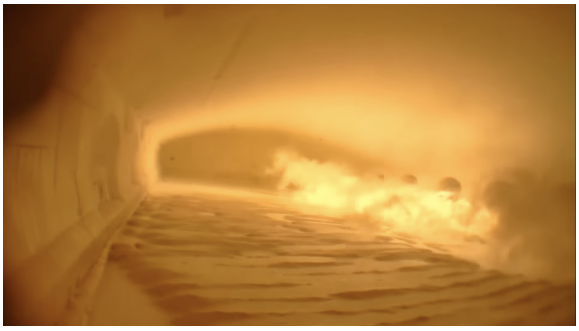
**Figure 3.2** – Views from the raw material preparation. From Saint-Gobain, 2021

### 3.1.2 Furnace

This part of the process is the main focus of this work. The batch is laid down at the entrance of the furnace, called the dog house, and then pushed inside of it. The objective of the furnace is to melt the glass by achieving a temperature between 1550°C and 1600°C. To do so, the glass is heated through the use of multiple gas burners (although more modern furnaces can use electric heating, through resistances, to lower carbon gas emissions and relieve the need for natural gas) which are placed above the glass bath, on both sides of the furnace, parallel to the stream of the molten bath. The burners alternate their activation every twenty minutes from side to side, allowing one side to burn gas while the other side allows for the extraction of the combustion gases. Beneath the flames of the burners, the batch melts and slowly flows downstream. The quality of the glass relies on the batch reaching the desired temperature (between 1550°C and 1600°C depending on specifications) as homogeneously as possible. There are multiple complex phenomena happening inside the bath: convection cells formed due to the temperature differences between the bottom and the top of the batch, chemical reactions happening at different rates, and the fluid displacements themselves, are all sources of inhomogeneity in the bath. After being exposed to the burners, the molten glass slowly flows toward the furnace exit; as it does so, it cools down from around 1400°C to 1100°C through the use of cool air. This is done to reach the correct viscosity for the rest of the process. It is important to note that it is thus possible to overheat the glass, and then have to cool it more, while the opposite is not true: it is not possible to reheat it if it had been underheated. Overheating leads to overconsumption of energy, while underheating leads to a potentially reduced glass quality, which most likely means having to scrap this glass and start the process again.



(**a**) Representation of the furnace

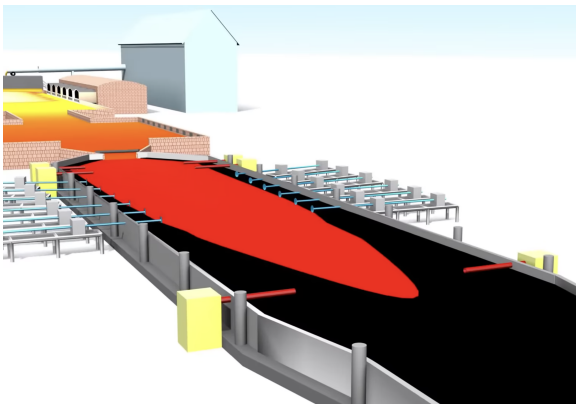


(**b**) View from the inside of the furnace

**Figure 3.3** – Side-by-side views of the furnace. From Saint-Gobain, 2021

### 3.1.3 Float Bath

After exiting the furnace, the molten glass ribbon is poured over a bath of molten tin. As tin is denser than the molten glass, the glass floats and spreads over the bath and forms a perfectly flat surface.



(**a**) Representation of the glass ribbon going through the float bath



(**b**) View of the solid glass (on top) exiting the tin bath (below)

**Figure 3.4** – Side-by-side views of the float bath. From Saint-Gobain, 2021

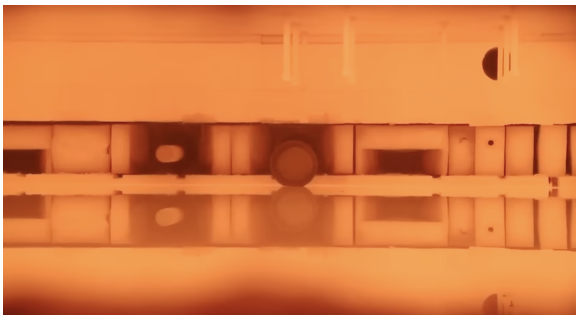
In order to control the thickness and width of the glass being formed, the flow speed of the ribbon has to be controlled. To do so, top-rolls machines are used on the edges of the glass ribbon to control its speed. This part of the process requires constant monitoring and control to adjust to the desired width and thickness of the glass sheet. After passing through the float bath, the glass has cooled down to around 600°C, allowing it to become solid.

### 3.1.4 Lehr

Once the glass is solid enough to be conveyed on rollers, it goes through the Lehr. Inside it, it is first carefully cooled down from 600°C to 480°C to anneal it, preventing the formation of



(**a**) Front view of a top-roll



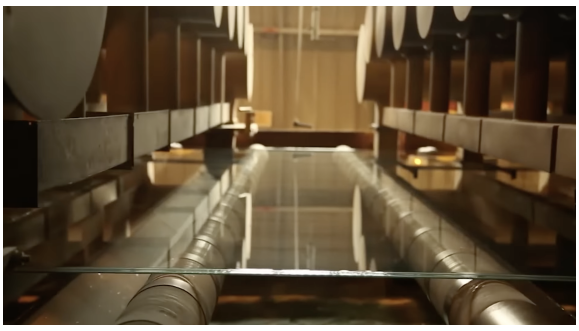
(**b**) Side view of the same top-roll

**Figure 3.5** – Front and side view of a top-roll. From Saint-Gobain, 2021

residual stress inside the glass. This allows the glass to reach the desired mechanical properties in order to be cut. Then it can be cooled down with less precision to reach temperatures of around 50°C to 80°C.



(**a**) Side view of the annealing section

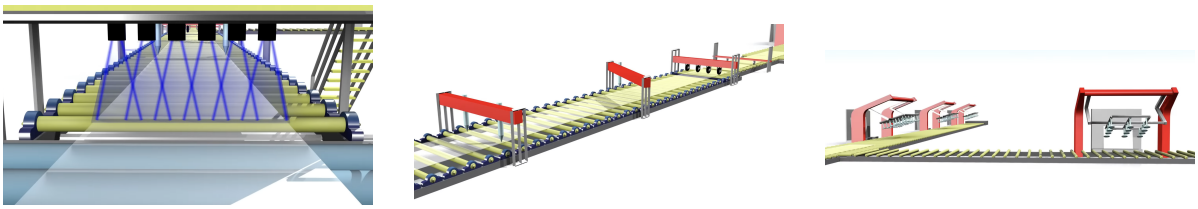


(**b**) Side view of the cooling section

**Figure 3.6** – Side view of the lehr, measuring from 150m to 200m long. From Saint-Gobain, 2021

### 3.1.5 Cold End

Now that the glass is cooled down, it goes through 3 last steps before shipment. It first goes through a continuous optical quality control check, to identify the presence of potential defects in the glass. It is then cut to the required dimensions. To do so, the sides are cut by a first set of diamond-wheel cutters, and the desired length is achieved by a second set of cutters. The individual sheets are then separated in the breakage area just downstream by pushing the sheets, which dislocate at the transversal cutting lines, while the edge rollers push the side parts down, trimming them. All the scraps (edges and sheets not up to standards) are collected to be recycled into the process. Once the sheets are cut, they are then removed from the line and stacked for transport.



(**a**) Quality control area (**b**) Cutting area (**c**) Stacking area

**Figure 3.7** – Views from cold end. From Saint-Gobain, 2021

## 3.2 Furnace Control and Its Current Limitations

In this work, we will primarily focus on the control of the temperature inside the melting furnace. As presented before, in the furnace the raw materials are heated to temperatures between 1550°C and 1600°C in order to melt them uniformly into molten glass. Not reaching these temperatures, or having some part of the batch not reach them, will lead to quality issues. Conversely, overheating, while not impacting quality, will lead to overconsumption of energy. While overconsumption seems like the lesser of two evils, especially since substandard glass will be scrapped and recycled, thus consuming additional energy anyway, it is important to note that at the temperatures at which the furnace operates, a difference of one degree represents a considerable amount of energy.

### 3.2.1 Scale of the Process and Cost of Imprecision

The float glass melting process is a large-scale, continuous industrial operation. A typical float line processes several hundred tonnes of glass per day and runs uninterrupted for campaigns lasting several years, as the cost of shutting down and restarting the furnace makes interruptions economically prohibitive. Sustaining temperatures above 1550°C around the clock requires substantial and continuous fuel consumption, making energy one of the dominant operating costs of the process. At this industrial scale, any systematic deviation from the optimal operating temperature accumulates over time: sustained over-temperature operation wastes energy continuously over the duration of the campaign, while under-temperature risks quality defects that lead to scrapped production and additional energy expenditure in recycling. Precise temperature setpoint tracking is therefore not only a technical requirement but a direct economic and ecological lever, which justifies investing in more capable control strategies beyond conventional approaches.

### 3.2.2 PID for Temperature Control

Currently, the temperature control is achieved through manual control, PID controllers (with various add-ons such as MPC), or a mixture of both. Manual control is based on years of experience of skilled operators; however, it can still be prone to errors or imprecision. PID (Ziegler and Nichols, 1942) controllers are a widely used and proven family of controllers, their role is setpoint tracking, and they can be tuned by modifying their parameters (three parameters in the simplest version of a PID). They are usually coupled with MPC (García, Prett, and Morari, 1989), which uses a model of the system dynamics to anticipate the future fluctuations and adapt the controller parameters. The current approach already yields satisfactory results, allowing for the production of quality glass. However, it is still not perfectly accurate, and as pointed out before, even a slight improvement in precision would yield significant results. PID and MPC are limited by the number of parameters, our ability to tune them, and the model

precision for MPC. Finding a better control solution, albeit a more complex and costly one, could be beneficial to go beyond the current capabilities of PID for this industrial application and other applications of the same family.

## 3.3 Why Reinforcement Learning?

In order to overcome the limitations of PID/MPC, we would want to try a method that allows for efficient and automatic tuning of a larger number of parameters, while allowing us to leverage more signal from the system and still allowing us to leverage expert knowledge (operators and current control methods). Reinforcement Learning is a textbook solution to this sequential decision-making problem. While not yet a competitor to PID in this type of application, limited by its far greater complexity and its much shorter usage history, we want to explore its use in this setting as we believe that in practice it would allow for better control at the expense of a more complex tuning process. As our understanding of the coupling of artificial neural networks with reinforcement learning is still limited, the performance or complexity of using reinforcement learning for this specific use case cannot be established without experimentation. While RL could offer better control strategies, these strategies may also not be realistically obtained with our current methods and computing power due to the need for a large amount of data or training time. However, some reinforcement learning applications drive us toward thinking that this type of problem is within the realm of possibility, at least in the near future.

While not being an off-the-shelf solution, the increasing number and complexity of applications of RL to process control is a good sign of the capability of RL to tackle a larger array of these applications.

## 3.4 Modeling the Furnace as a Control Problem

### 3.4.1 The Furnace MDP

In order to apply RL to the control of the temperature inside the furnace, we must first properly define the problem at hand. To do so we will define the Markov Decision Process (MDP) representing the furnace temperature control problem. To control the temperature, the PID controllers and operators base their decisions on a set of measures of the temperatures inside the furnace. This temperature is measured at multiple points, by sensors placed outside of the actual furnace for protection, thus measuring through the roof of the furnace. The temperature is controlled through the modulation of the quantity of gas injected into the burners which then inject more or less energy into the bath through the flames they produce. The goal of the process is to keep the temperature in the furnace as close to the setpoint as possible at all times, while preferring overheating to underheating. The temperature should also never exit some safety boundaries to prevent the degradation of the production equipment.

We will now proceed to translate that into an MDP:

- The state space $\mathcal{S}$ is hard to perfectly define as it needs to represent all the phenomena happening at every point inside the furnace, which would require an infinite number of dimensions. While a subset of those points may be enough to retain the Markov property, this subset would most likely still be of very high dimension. $\mathcal{S} \subset \mathbb{R}^k$ with $k \in \mathbb{N} \gg 1$. In practice, we consider this state space unknown.
- The observation space $\mathcal{O}$ is comprised of the different temperature measurements (temperature sensors distributed on the roof of the furnace). $\mathcal{O} \subset \mathbb{R}^l$, with $l \in \mathbb{N}$ the number of temperature sensors.
- The action space $\mathcal{A}$ is the selected level of power of the burners (between 0 and 100% of available power). $\mathcal{A} = [0, 1]$. All burners are controlled using the same power level.
- The transition function $P : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is once again too hard to define explicitly, as it contains the different physical phenomena at play in the bath. We treat $P$ as deterministic throughout this thesis; the stochasticity of the environment arises from partial observability and unmodeled disturbances rather than from an intrinsically stochastic dynamics. We do know some properties of this function, it is smooth and bounded. We do understand the different phenomena at play independently to some degree. We can model each of these phenomena, independently, to some extent. As for the state space, we will assume this function to be unknown.
- The reward function $R : \mathcal{O} \to \mathcal{R} \subset \mathbb{R}^-$, with $\mathcal{R}$ the reward space, is a measure of the error between the temperature setpoint and the current temperature measurements (a reward of 0 being a perfect match, and negative values indicating the magnitude of the discrepancy).

### 3.4.2 Key Properties

In addition to the properties of a basic MDP, the furnace presents additional key properties that we want to take into account to allow for the finding of control strategies tailored to these issues.

- **Partial-observability** (Kaelbling, Littman, and Cassandra, 1998). As presented in the state-space definition, the state of the system is widely complex. In practice the current furnace instrumentation is restricted to a discrete number of temperature observations with several sensors. As the sensors are not perfect by nature, and they are measuring the temperature through the roof of the furnace, we are only dealing with a noisy and partial observation of the system state.

- **Non-stationarity**. As presented in Sections 3.1.1 and 3.1.2, there are multiple sources of external perturbations that are outside of the system that may evolve over time. The composition of the batch[2] may evolve depending on the product being produced, its humidity may vary depending on the outside humidity and storage conditions. Both factors changing how much energy is needed to heat and therefore the system dynamics. The outside temperature will also vary depending on the time of day, the time of year, and the weather at the location. This will impact the temperature of the air used in the burners, as well as the temperature outside the furnace and therefore the temperature loss of the furnace. While this is, strictly speaking, a form of partial observability, we still prefer to list it separately as it is of a different nature than the previously mentioned property.
- **Real physics**. The system we study is an actual physical system, obeying the laws of physics. This ensures a number of properties needing to be respected, continuity, derivability and thus smoothness. This is by contrast to video games, which can abide to arbitrary laws, without guarantees for the aforementioned properties. This also means that our understanding of the underlying physics can most likely be used as useful knowledge for the identification of a good control policy. For example, we know that adding energy through the burners will positively impact the quantity of energy in the furnace, while reducing their input will do the opposite. We also know that the furnace will cool down if no gas is burned, as it cannot raise its temperature by itself.
- **Setpoint tracking**. The goal of temperature control is to maintain the furnace temperature as close as possible to the target temperature (which we assume is known, as it was set as part of the task). This means that the maximal instantaneous reward is obtained when being exactly at the target; and that the perfect average reward is this maximal instantaneous reward. Contrary to other tasks, like MuJoCo suite, that want to maximize an unbounded score[3], in our setting we are aware of how close we are to optimality (but not necessarily the optimal policy, as we do not know if a perfect score is possible). Additionally, this means we know the subset of the state space we wish to stay in (all states which have the correct temperature). Contrary to goal-MDPs, where we wish to reach a state and end the task there, here we wish to maintain this state for as long as possible. A corollary of this setting is also an access to dense rewards (Andrychowicz, Wolski, et al., 2017), as the desired states are clearly identified. While dense rewards are not exclusive to this type of task, in setpoint tracking, the metric we wish to maximize is also easily accessible.

[2]Here batch refers to the materials, not the machine learning sense of batches of data
[3]while the score is bounded by the maximal velocity the robot can achieve, we do not know the bound

- **Stability around the setpoint**. A specificity of the class of problem we study is the fact that some subset of the states of our system are stable under constant policies[4]. In the furnace, if one raises the power input from $P_1$ to $P_2 > P_1$, the temperature will rise from $T_1$ to $T_2 > T_1 \ll \infty$. In other words, to every power input, corresponds a stable temperature that the furnace will rise (or decrease) to and stabilize at. This property is in contrast to other setpoint tracking tasks, like CartPole, where when considering a constant input, only a specific velocity will maintain the pole close to equilibrium, while all the others will inevitably lead to the pole either falling left or right. Another analogy can be seen in the control of a car. For speed control, if the driver increases the amount of pressure on the throttle pedal, the car will increase its speed until it reaches some new stable velocity, while if the driver increases the rotation of the steering wheel the car will keep turning and never stabilize at a new angle (assuming you cannot crash into something first). This property stability can be useful for control as it gives some signal about the states which the action can reach if given enough time and no perturbations.
- **Delay** (Ramstedt and Pal, 2019). In typical real systems, there is always a notion of delay between the action and our observations of the system response to this action. Depending on the system, this delay can be negligible, or have a significant impact on learning a policy. In the case of the furnace, there is a significant delay between a change in the burner power input and the observation of the full impact of this power step on the temperature of the furnace (several hours in practice). Phrased differently, some significant period of time following the power input change contains little to no signal regarding the effect of the action on the system, and thus cannot be used to measure its quality with respect to controlling the temperature. Delay is a subtype of partial-observability, as while the impact of actions on temperature is not instantaneous, the underlying real state of the system still upholds the Markov property and thus already contains the full impact of the action. However, even though delay is a sub-type of partial-observability, we believe it is important to consider this aspect individually when identifying solutions.

All these properties make up the class of problems we would like to model in order to represent both the glass furnace and a wide range of other industrial process applications. These properties are both constraints needing to be accounted for and making the task of finding an optimal control solution harder (e.g., partial observability, non-stationarity), and also properties that can be leveraged to make this search easier by restraining the type of solutions we explore (e.g., real physics, setpoint tracking).

[4]Policies which always select the same action with a probability of 1.

## 3.5 Interaction or Simulation

In order to use reinforcement learning to learn better control strategies for the furnace, we need to interact with the system. As presented above, the system is a massive furnace with very high financial and energy costs, while also posing high risks if errors were made. These downsides make using the actual furnace a safety risk without proper precautions. In addition, as reinforcement learning tends to be very data-hungry, training an RL algorithm could require on the order of millions of data points. Collecting those points from a single and expensive furnace would be slow and impractical. These safety, cost and speed constraints point to using a simulation of the furnace instead. Indeed, using a simulation allows us to rise above these issues by allowing for parallelization of computations for faster data collection, while also removing the need for an expensive physical system and using cheaper computations. However, this is only true if we are capable of simulating the furnace precisely enough for the control strategies learned to transfer easily to the actual system, and that the simulation can actually be simulated at least as fast as the physical system at a realistic cost.

Due to the complexity of the phenomena happening in the furnace, building an accurate simulation is a very hard task. Current simulations of the furnace are limited to a single timestep and do not model non-stationary phenomena. We think that learning a model from historical data would have been too time-consuming and might be imprecise in the end, so we discarded this idea in favor of an alternative solution, using a proxy model.

## 3.6 Identification of a Proxy Problem

The goal of the proxy problem is to have properties as similar as possible to the original problem, while allowing for much simpler simulation. The proxy we are looking for would ideally satisfy additional constraints rather than just matching the properties listed above. We could try and augment an existing task with all the properties listed above, for example adding all these properties to the traditional CartPole environment (A. G. Barto, Sutton, and C. W. Anderson, 1983). However, some of these properties may not naturally appear in these environments (e.g., there is no natural delay in CartPole, there is no infinite horizon in the classical setting). While one could easily still understand the task with some of these properties added, even if they feel artificial (e.g., just adding some delay in the inputs to CartPole), some other properties will require profoundly modifying the task (e.g., adding stability to CartPole). As a consequence, we would prefer a problem that naturally presents all the key properties listed above (see 3.4.2). After numerous attempts to identify such an environment, we finally identified a good candidate task: airplane altitude control.

### 3.6.1 Airplane Altitude Control

In this task, the goal is to control a 2D jet aircraft which can only evolve in the x-z plane and rotate around the y-axis (i.e., from the pilot's perspective, it can climb or descend, but cannot go left or right) to reach and maintain a target altitude. This can be achieved by controlling the thrust of the jet engines and the pitch of the aircraft.

An MDP for the airplane altitude control task could be as follows:

- The state space $\mathcal{S}$ would require knowledge of the state (velocity, pressure, temperature) of every air particle around and in front of the aircraft, as well as the velocity (and angular velocity) and position (and angular position) of the aircraft.
- The observation space $\mathcal{O}$ is the subset of sensors we actually have access to, this includes the velocity of the aircraft with respect to the apparent wind(which we assume is the velocity of the center of gravity of the aircraft), the vertical velocity (climb rate), the horizontal velocity with respect to the ground (ground speed), the vertical position (its altitude), and the angular position with respect to the ground (its pitch).
- The action space $\mathcal{A}$ is composed of two dimensions, one for the engine thrust (between 0 and 100% of available power), and one for the stick position controlling the pitch (between -15° and 15° of stick angle).
- The transition function $P : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ (deterministic, consistent with the convention established in §3.1.2). Here, contrary to the furnace, we have access to model equations that accurately predict the system in the form of aerodynamic force formulas. We will consider these formulas exact. The details of these computations can be seen in Appendix B.
- The reward function $R : \mathcal{O} \to \mathcal{R} \subset \mathbb{R}^-$, with $\mathcal{R}$ the reward space. The reward would be proportional to some measure of the discrepancy between the current altitude and the target altitude. In addition, and unlike the furnace, we could also add a large negative reward if the plane crashes into the ground.

We will now go through the key properties of the furnace and see how the airplane environment compares to them.

- **Partial-observability**. Just like the furnace, it is not possible to observe the whole system. It would require knowing the velocity, pressure and temperature of every air particle around the airplane. However, unlike the furnace, in practice only knowing the velocity and position of the airplane is largely enough to get a good control policy. In other words, the signal contained in the available observations in the airplane environment is far richer than the one contained in the available observations for the furnace. So, although both systems are partially observable, it is a much bigger issue for the furnace.

However, this can be emphasized in the airplane task by introducing noise or removing some observations to make the task closer to the furnace setting.

- **Non-stationarity**. The dynamics affecting the aircraft are heavily dependent on the characteristics of the surrounding air. Similarly to the non-stationarity introduced by the variation in raw material humidity and composition, the velocity, pressure and temperature of the air around the aircraft will affect the aircraft trajectory. Thus, from the aircraft's perspective, the aerodynamics will seem to evolve over time.
- **Real physics**. As for any real system, the aircraft obeys physical laws. In this case, the laws of aerodynamics apply.
- **Setpoint tracking**. In this altitude control task, the goal is to reach and stabilize the aircraft at the desired altitude. The goal is to automate some of the aircraft control to relieve the pilot's mental load and to minimize constant inputs.
- **Stability around the setpoint**. Similarly to the furnace, increasing the power input will have the plane climb and stabilize at a higher altitude, while decreasing the power input will have the plane descend and stabilize at a lower altitude. This is due to the power increase leading to an increase in velocity, which turns into an increase of lift that makes the plane climb. As the plane climbs the air density decreases, which reduces both the lift and the engine efficiency. The reduced velocity reduces the lift, combined with the already reduced lift effect from air density, this reaches a point where the lift is once again only enough to counteract gravity, leading to a stable altitude. The same goes for a descent, where air density would increase as the plane descends. To some extent, this phenomenon is also applicable to the pitch inputs; however, past some point the plane will either stall and fall if the pitch is too high, or just descend and crash if the pitch is too low. This part differs from the furnace; however, if the aircraft does not stay in these situations for too long, this can be avoided.
- **Delay**. When a change of power input is applied, the engines will start increasing their rotational speed. However, this increase is not instantaneous and will take a few seconds to stabilize fully. As a consequence, there is some delay before the impact of actions can be observed on the aircraft. There is no significant delay regarding pitch control (as the time between the action, and the transmission of the signal to the rudders can be neglected).

Unlike furnace physics, aircraft physics are well known and relatively easy to model accurately. Although more accurate simulations could be done using CFD (Computational Fluid Dynamics) simulations, the accuracy given by the use of aerodynamic equations is enough for our application. The simplicity of these equations allows us to implement a simulation of this task that will run in a time compatible with large RL experiments.

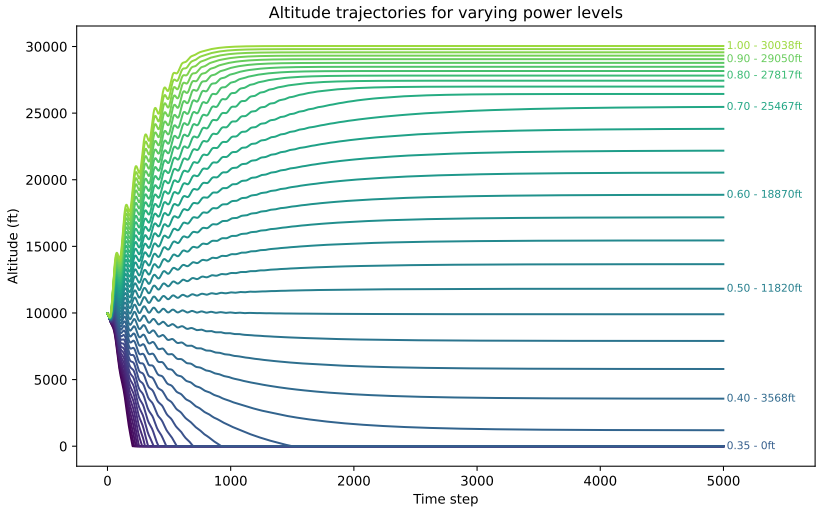



**Figure 3.8** – Example altitude trajectories of the aircraft simulation under different power inputs, illustrating the stability around the setpoint (Berthelot, 2025b).

## 3.7 Implementation

In order to run our experiments, we implemented a 2D aircraft simulation of an Airbus A320, with a focus on simulation speed using JAX (Bradbury et al., 2018). This environment is open-source and available as part of the Target-Gym suite (Berthelot, 2025b). For more details, refer to Section 7.3, where we delve into the implementation details.

## 3.8 Conclusion

This chapter established the problem that motivates the rest of this thesis. Temperature control inside a float glass furnace is a typical instance of industrial process control: a real physical system where the goal is to track a setpoint as precisely as possible, under partial observability, non-stationarity, significant actuation delay, and without any safe mechanism for direct experimentation. The economic and environmental stakes make even marginal improvements in control precision valuable. Current PID/MPC controllers are mature and well-tuned, which sets a high bar: any RL-based solution must not merely learn to control the system, but do so better than methods refined over decades of industrial use.

Because the furnace is too expensive and safety-critical for direct RL experimentation, we identified the 2D aircraft altitude control task as a proxy. The aircraft shares all of the furnace's key structural properties, including partial observability, non-stationary dynamics, real physics, setpoint tracking with stable equilibria, and actuation delay, while being cheap to simulate

accurately. This makes it a principled and practical stand-in for the algorithmic work that follows.

With the problem formalized and a simulation platform in hand, the next step is to survey the landscape of existing approaches. Chapter 4 reviews the three bodies of literature that converge in this thesis: applications of RL to industrial process control, methods for improving sample efficiency, and techniques for robust value estimation.

# Chapter 4

# Literature Review

> "The ancients had only the books which they themselves wrote, but we have all their books and moreover all those which have been written from the beginning until our time.… Hence we are like a dwarf perched on the shoulders of a giant. The former sees further than the giant, not because of his own stature, but because of the stature of his bearer. Similarly, we [moderns] see more than the ancients, because our writings, modest as they are, are added to their great works."
>
> Glosae super Priscianum — William of Conches (c. 1123)

In this chapter, we will take a look at the relevant literature for this thesis. To do so, we will cover the different fields that merge into our study. Starting from the thesis title "Robust and Efficient Reinforcement Learning for Industrial Process Control," we can identify the following subjects of interest: Robust RL, Efficient RL and RL for Industrial Process Control. We will study the literature regarding each of these subjects.

## Contents


**4.1 Reinforcement Learning for Industrial Process Control** . . . . . 44
**4.2 Sample-Efficient Reinforcement Learning** . . . . . 49
**4.3 Robust Value Estimation in Deep RL** . . . . . 56
**4.4 Conclusion** . . . . . 58

## 4.1 Reinforcement Learning for Industrial Process Control

As mentioned in Section 3.3, where we justify our interest in using RL for this problem, the literature offers some examples of applying reinforcement learning to industrial process control. Here, we will dive deeper into this literature to understand the successful applications of deep RL to real-life complex dynamical systems, what they share with the furnace, and which difficulties they reveal for our own case. We will also cover two examples of the RL community taking interest in industrial process control projects: the RLCore (RLcore, 2026) company, and the Process-Control Gym benchmark (Bloor, Torraca, et al., 2026).

### 4.1.1 Traditional Control Methods

The class of problems we are interested in, where the goal is setpoint tracking, is typically addressed through the use of PID (Ziegler and Nichols, 1942), Linear Quadratic Regulator (LQR, R. E. Bellman (1957)) (Kalman, 1960) or MPC controllers.

The latter method, more general than the others, is usually preferred to tackle more complex systems as it allows for explicit constraint consideration. MPC methods use a model of the system to predict the impact of sequences of actions. The method involves solving an optimization problem at each step to maximize the return of this sequence of actions over a given horizon. The first action of that optimal sequence is then applied to the system, before observing the new state and restarting the whole process again. It is important to note that the optimization problem is solved over a limited horizon; hence, the objective is myopic and does not take into account long-term effects. While the horizon could be increased to better represent long-term effects, that would also increase the compounding errors of the model.

While widely used and potent, MPC is not perfect and has shortcomings that limit its efficiency in complex cases. O. Dogru et al. (2024) list the weaknesses of a traditional MPC approach as:

- Relying on a model of the process. This model may be hard to design depending on the system. This also translates to being limited by the accuracy of the model.
- "General" MPC is limited to linear or quadratic objective functions.
- It does not generally offer an adaptive form.

MPC evolutions that try to solve these weaknesses have naturally been proposed:

- Robust MPC, also referred to as Tubular MPC (Oshin, Almubarak, and Theodorou, 2024), tries to limit the impact of model errors and environmental noise. It does so by using the traditional MPC approach (denoted nominal MPC) of imagining the trajectory under no noise or errors, and then also imagining multiple variations of this trajectory under perturbations (denoted ancillary MPC), forming some bounds on where the true states will likely lie.

- Stochastic MPC (Mark and Liu, 2020) is an evolution of Robust-MPC. Where Robust-MPCs have a "worst-case scenario" approach, Stochastic MPC exploits some model of the disturbances to allow trade-offs between satisfying the constraints and controller performance.

Combining control methods with RL has been studied through various approaches. The majority of those methods revolve around the use of RL to determine the parameters of the controller (D. P. Bertsekas, 2024; Sharma, Suttle, and Sivaranjani, 2025; Oguzhan Dogru et al., 2022; Char and Schneider, 2023). As these parameters are classically tuned using heuristics, having RL guide this tuning using the error signal is a potential solution. In this framework the policy is the controller, and its weights are the parameters of the controller. Due to its connection with traditional control methods, this approach has logically seen practical applications: grinding mills (van Niekerk, le Roux, and Craig, 2025), mobile robot control (Yu et al., 2025), and tank systems (Oguzhan Dogru et al., 2022). In addition, implementing an MPC approach as part of the RL algorithm itself has also been done through the TD-MPC (Hansen, Su, and X. Wang, 2022) and TD-MPC2 (Hansen, Su, and X. Wang, 2024) algorithms. Due to their importance in the model-based RL literature we will discuss them in the next section.

### 4.1.2 Deep RL for Tokamak Control

Before providing an overview of the most relevant aspects of the literature regarding our work, we will first examine in detail the most influential piece of literature for our work: Tokamak nuclear reactor control (Degrave et al., 2022). In this work, the authors use reinforcement learning to control the shape of plasma inside a tokamak reactor, an impressive feat given the instability of plasma, the frequency at which actions must be selected (10kHz), as well as the cost and complexity of collecting transitions from such a complex system.

**Algorithm and architecture** To achieve this, the authors use the Maximum A-Posteriori Policy Optimization (MPO, Abdolmaleki et al. (2018)) algorithm, an actor-critic algorithm. The actor and critic networks both use Long Short-Term Memory (LSTM) layers to account for partial observability, and note that using asymmetric networks (the critic networks being larger than the actor network) leads to better policies. It is important to understand that under their constraint of a fast policy inference (10kHz) it is crucial to keep the networks as small as possible while maintaining performance. As the critic is only used during training, and not for inference, it can be larger as it will not impact inference time.

**Mass parallelism** An important aspect of their approach is the use of 5000 actor networks collecting data from independent simulations. By independent simulations, we mean that they can be initialized with different initial states, and that they will evolve independently of

others, notably they may not terminate at the same time. As seen in Rudin et al. (2022), the idea to collect from a large number of independent environments using parallel actor networks seems to lead to interesting benefits and emerging properties. The first and most obvious is that by collecting experience in parallel, this reduces the collection time and thus makes learning faster. In environments where the time to collect a timestep (simulated or real) is a significant part of a learning step (experience collection and using the experience to update the policy), parallelization of the experience collection has a potential for massive improvement of the algorithm speed. The second and less obvious advantage of using a large number of independent environments is the introduction of more variety into the collected experience. This is achieved naturally by using multiple independent environments with different randomly sampled initial states, which will lead to different trajectories. However, the diversity can also be artificially enhanced by introducing domain randomization (small perturbations of the environment transition and/or observations, as in Tobin et al. (2017)) in every environment to prepare the agent for slightly different situations, most likely allowing the agent to become more robust to these variations. We discuss this subject in more detail in Chapter 6.

**Simulator** The achievements of this paper are two-fold: the method to obtain the policy from experience is only half the work as obtaining experience to learn from is the other and necessary part of the work. Just like the furnace, building a simulation for the system we wish to control can be harder than finding a good policy. In this work the authors have managed to create a simulator (Citrin et al., 2024) capable of being accurate enough while allowing for a simulation time compatible with RL (the authors report training times from 1 to 3 days). We believe this article perfectly emphasizes the fact that for real-life applications, framing and formulating the RL problem is as important as the method to solve it. Due to the importance of nuclear fusion, this work is of great interest for the community, however, the complexity of the system is an important barrier for one wishing to develop algorithms for this purpose. The Torax (Citrin et al., 2024) simulator developed as part of this work is most likely inaccessible without a keen understanding of plasma physics; however, recently a reinforcement learning environment, gym-torax (Mouchamps et al., 2025) has been created out of it and should allow for easier entry for the RL community.

### 4.1.3 Other RL Applications for Industrial Process Control

RLcore (RLcore, 2026) is a company that states that its goal is to apply RL to industrial control, citing examples of active deployment in Water and Wastewater. Their first use-case (RLcore, 2024a) is the optimization of water treatment ultrafiltration, which translates into the real-time control of coagulant dosing and filter backwashing. They claim that their RL solutions led to a "13% reduction in chemical usage over 5 days during high-dosing event" and "14.6% reduction in backwash wastewater", both compared to the existing manual control baseline. The second

documented use-case (RLcore, 2024b) is the reduction of wastewater treatment emissions by using RL to improve $H_2S$ scrubber efficiency. For this use-case, they state that using RL instead of "traditional control methods" led to process efficiency going from 80% to 95% while halving the process costs.

Although the first use-case would have benefited from comparison to a simple tuned PID controller, to verify that reinforcement learning was indeed justified, these examples are still greatly valuable as examples of successful applications of reinforcement learning on real-life systems with real deployment and benefits. Similarly to the Tokamak example, these examples also show that RL is only half of the work in real-world applications. Deployment, sensors, and data collection are as much, if not more, of a hurdle for the real-life applications of RL.

Another work that takes an interest in the use of RL for industrial applications is the collection of benchmark environments named Process Control Gym (PC-gym Bloor, Torraca, et al. (2026)). We believe that one of the reasons that RL is not currently tailored for industrial applications is the lack of industrial application environments available for the RL community. Most RL researchers are keen on developing better algorithms; however, they can only make them better for specific applications if they have access to the corresponding environments. It thus requires the connection of people knowing both RL and processes to build those environments in the first place (in the same way that it takes people knowing robotics and RL to build robotic environments) and start the community. PC-gym proposes a suite of environments, simulating various chemical engineering processes, that are built for RL algorithms benchmarking. As discussed in Section 3.4, we believe a wide fraction of industrial processes share a large majority of the key properties we identified in the furnace. PC-gym environments are part of these tasks. In addition, PC-gym allows for comparison with Nonlinear Model Predictive Control (NMPC) oracles, a highly practical tool that avoids shifting the burden of designing these controllers to RL practitioners. This also highlights an important point of using RL for industrial process control: the comparison to traditional control methods like PID or MPC. As the goal is to someday be able to find methods more efficient than the traditional ones, it is important to compare the performance of RL algorithms to these traditional methods. However, it is also important to remember that, as RL will most likely have worse results than the simpler traditional methods at first, RL does not have the decades of thorough usage and experience in the industry that led to the refined traditional methods we know today…yet!

Finally, pure RL approaches have also been applied to various industrial processes. A wide range of these applications are problem classes beyond setpoint tracking. In these applications, a controller (PID, LQR or MPC) could not solve the task, usually due to the lack of dense rewards. These applications mostly revolve around industrial robots, for example: Insertion tasks (Schoettler et al., 2020), Additive Manufacturing (Ali et al., 2025), Cyber Physical Production Systems (Kuhnle and Lanza, 2019). However, there are also applications of RL to process

control and setpoint tracking: water treatment through anaerobic digestion system dilution rate control using Deep Deterministic Policy Gradient (DDPG, T. P. Lillicrap et al. (2016)) (Mendiola-Rodriguez and Ricardez-Sandoval, 2022), volumetric flow control in chemical reactor (Powell, Machalek, and Quah, 2020) using A2C (the synchronous variant of Asynchronous Advantage Actor-Critic (A3C, Volodymyr Mnih, Badia, et al. (2016))), Integrated energy system cost optimization under uncertainty (Zhang et al., 2019) using PPO, or fluid mixing in various industrial processes (Konishi, Inubushi, and Goto, 2022) using DQN.

**Industrial perspective**

The literature already demonstrates that RL can successfully control genuinely complex industrial systems: from plasma shape in a Tokamak reactor to coagulant dosing in water treatment and volumetric flow in chemical reactors. These results confirm that the problem class targeted in this thesis, continuous setpoint tracking under challenging dynamics, is tractable with deep RL, and that the gap between academic benchmarks and real industrial deployment is closable.

### 4.1.4 Conclusion

We have seen that there are already a few examples of RL being applied to industrial process control. These examples show that using pure RL approaches, or a mix of RL and traditional controllers, can outperform traditional controller approaches, and thus makes RL a promising method to deal with this class of problems. However, we also note that in pure RL approaches, the choice of the used algorithm over other algorithms is usually not strongly justified. Overall, this leaves us with a feeling of using already existing approaches and testing them on industrial problems, but not trying to create methods suited for industrial problems. With the exception of the work of Degrave et al. (2022) on the Tokamak, which had significant modifications to the hyperparameters when compared to the original algorithm (but lacked an explanation as to why and how they identified the new values), all other surveyed work use an existing algorithm without a proper justification. While these are still great proofs that RL can indeed be a solution, they lack a structured approach to identify the proper methodology for applying RL to an industrial process control problem. As we do not believe that the RL community is inherently opposed to industrial tasks, we instead prefer to put the blame on the lack of industrial tasks available to the RL community. We will address this issue in Section 7.3. As building such tasks requires knowledge about said processes, this seems to lead to the fact that people who end up experimenting with RL on an applied problem have significant knowledge about the problem, but less knowledge about RL. People that work on applications have the skills and knowledge to model the complex systems they work with into equations. It is thus the duty of the RL community to provide algorithms that are suited for these problems. As for

any research problem, there is of course a bias toward positive results in the literature, which may obscure the problems for which RL was unsuccessful.

## 4.2 Sample-Efficient Reinforcement Learning

Efficiency in RL can be interpreted in three ways: Time-Efficiency (How long will it take to reach a given level of performance), Sample-Efficiency (Kearns and Singh, 1998; Strehl and Littman, 2008) (How many samples are needed to reach a given level of performance), and asymptotic performance (the score of the best policy asymptotically obtained by the algorithm). While performance can be part of a trade-off with sample efficiency or time efficiency, it is still the main target of any RL algorithm. As such, we believe that a "performance"-oriented RL algorithm is a pleonasm. Therefore, we will speak about the performance of algorithms without having performance be a class of RL algorithm. It is also important to note that time-efficiency, sample-efficiency or performance are all relative properties that are strongly tied to the task at hand and the chosen hyperparameters, and not absolute properties of the algorithm.

While a perfect algorithm would have both a high sample and time efficiency, in practice it is more often than not a trade-off between the two types. As a consequence, depending on the setting, one type of efficiency may be preferred over the other. For example, in practical applications where data is sparse and expensive (e.g., learning from a real system without access to a simulator), sample efficiency will be preferred. In that situation time efficiency would likely be pointless as a faster algorithm would still have to wait for the collection of new samples; however, sample efficiency is key as it would lessen the overall training time and cost. On the opposite side of the spectrum, if sample collection is fast and cheap (e.g., access to an efficient simulator) then sample efficiency is less important as long as collecting new samples is faster than learning more from the existing ones. The distinction is important, as based on the industrial examples we reviewed previously, approaches seem to favor time-efficiency over sample-efficiency. Nonetheless, a large part of the literature is still taking interest in tasks where learning is directly done on the real system. This large portion of the literature shows that learning from few samples (most likely interacting with the real system) is still a huge challenge for RL (Kalashnikov et al., 2018; Dulac-Arnold et al., 2021; Tang et al., 2024).

Although learning from a simulator comes with its difficulties (designing the simulator and bridging the gap between simulation and reality), it can still be beneficial, even with the intent of running on a real system. It is important to improve algorithms in both efficiency types if possible. Even if sample collection using simulators is much more practical and faster than learning from real systems, training time is still a significant limit to the performance of RL algorithms. It is important to note that while sample-efficiency does not care about time, only the number of samples collected, time efficiency is affected by the number of samples collected. The more samples are collected, the more time needed to process them. As a consequence,

even when dealing with fast sample collection, increasing the speed of an algorithm at the expense of sample efficiency is only viable if it reduces the overall training time. Thus, it is still important to look for time-efficient ways to improve sample efficiency.

A sample-efficient algorithm will use fewer experience samples to learn a policy with the same or better performance than a less sample-efficient algorithm. To create sample-efficient algorithms, the goal is thus to learn as much as possible from every piece of experience.

Figure 4.1 provides an overview of the sample-efficient algorithms reviewed in this section, organized along two axes: how the model (if any) is used (columns) and the UTD ratio (y-axis). Two conclusions emerge from this map and will structure the rest of this section. First, high sample efficiency does not require a model. Second, model-based methods are not a monolithic family but differ fundamentally in how the model is used and in what it actually models: some methods learn a transition model over states, others learn a reward model, others learn both, and yet others operate entirely in a learned latent space rather than in the original observation space.

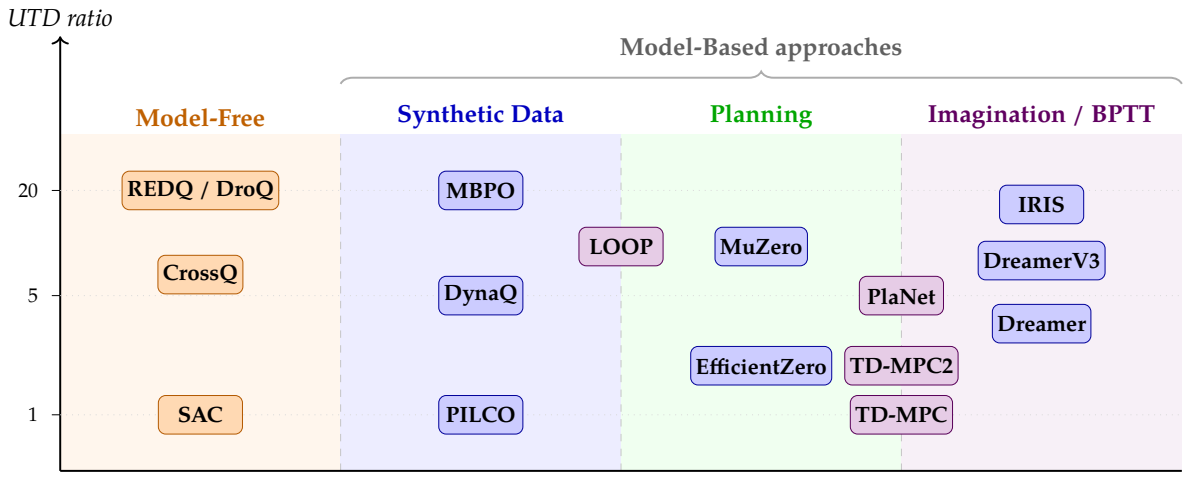



**Figure 4.1** – Taxonomy of sample-efficient RL algorithms organized by **how the model is used** (columns) and the **Update-To-Data (UTD) ratio** (y-axis, approximate log scale). Three mechanistically distinct model-based families are distinguished: **Synthetic Data** (the model generates fake transitions added to a replay buffer and consumed by a model-free update); **Planning** (the model is queried at decision time to search over action sequences via MCTS or CEM); and **Imagination/BPTT** (the policy is trained **through** full imagined trajectories in a learned latent space via backpropagation through time). Algorithms placed between two columns combine both mechanisms. Two key observations emerge: (i) high sample efficiency does not require a model (REDQ/DroQ at UTD=20); (ii) model-based methods largely achieve sample efficiency at UTD≈1, relying on model quality rather than update frequency. UTD values are for illustrating the ranking among algorithms rather than the exact values.

### 4.2.1 Model-Free Methods

A natural first approach to improving sample efficiency is to increase the number of gradient updates performed per environment step, the UTD ratio, without relying on any model of the

environment. The key challenge when increasing the UTD ratio is that neural networks tend to overfit to recently collected data and lose the ability to adapt when updated too frequently (see Section 4.3.3). The algorithms below address this challenge through modifications to the critic architecture or training procedure.

REDQ (X. Chen et al., 2021) introduces an ensemble of Q-networks; for every prediction, 2 critics are randomly sampled and used. Using a larger number of critics allows one to reduce the overestimation bias inherent to critics (Fujimoto, Hoof, and Meger, 2018; Hasselt, Guez, and D. Silver, 2016), and the random sampling allows one to split the numerous updates across a larger number of learners, to prevent overfitting. These modifications enable stable training at UTD=20. Dropout Q-Learning (DroQ, Hiraoka et al. (2021)) shows that adding dropout and layer normalization to the SAC critic is sufficient to stabilize high-UTD training, without resorting to a large number of critics.

In the same family of algorithms there is also CrossQ ( Bhatt et al. (2024)). While CrossQ falls back to an UTD of just 1 (like SAC), it still improves on sample efficiency. It achieves this by removing target networks, which introduce lag in learning. To compensate for the instability that the value target non-stationarity creates in learning (target networks being present for this very purpose in the first place), CrossQ uses cross-batch normalization and concatenated forward pass. By concatenating the current state and the next state (used as the target) to pass to the critic, batch-normalization statistics can adjust for both at the same time. To prevent the gradients from flowing back from the target prediction and create a moving target, a stop gradient operator is used for the targets. Using these tricks, CrossQ achieves a sample efficiency better than SAC, and on par with the larger UTD-methods like REDQ and DroQ, while reducing the computational cost.

These results show that **sample efficiency is not limited to model-based methods**. A model-free algorithm with a well-regularized critic can match or exceed the sample efficiency of many model-based approaches, without the added complexity of learning a world model.

### 4.2.2 The Model-Based Family Is Not Monolithic

Model-based methods are usually what comes to mind when thinking about increasing the sample efficiency of an algorithm. Model-based methods rely on an explicit model of the environment, allowing us to predict the impact of actions on the state[1] of the environment, and therefore 'imagine' possible trajectories. This allows us to learn from this synthetic experience as well as real experience, which reduces the amount of real samples needed for training. This is, of course, relying on the quality of the model.

[1] while most methods predict the state, some only predict an embedding of the state and not the state itself.

Model-based methods can be considered as old as model-free methods, as Dynamic Programming predates RL itself. However, the requirement for a perfect model of the environment is too conservative, making these methods mostly unused in practice. While Q-learning (Watkins, 1989; Watkins and Dayan, 1992), one of the most popular tabular RL methods, dates from 1989, the Dyna architecture and its Q-learning instantiation DynaQ (Sutton, 1991) (which is widely considered as the first explicit model-based algorithm) date from 1991. For a thorough survey on Model-Based Reinforcement Learning (MBRL), please refer to Moerland, Broekens, and Jonker (2020). In this thesis we will only focus on a subset of relevant algorithms.

It is important to note that model-based methods are not only useful for sample efficiency; they can also improve the asymptotic performance of the algorithm by improving planning capabilities. However, the opposite can also be true, as sometimes learning or using an inaccurate model can lead to the model biases polluting the policy and limiting its capabilities (Hansen, Su, and X. Wang, 2022; Ha and Schmidhuber, 2018).

When speaking about deep model-based methods, a key fact is that we are not exactly interested in an accurate model of the environment. In practice, we need a model of the environment that is accurate for representations of the states such that these representations lead to good policies. In other words, we want value-aware (in the sense of RL value functions) models as described in Farahmand, Barreto, and Nikovski (2017).

This difference is crucial; let us go through an example to understand why that is. Suppose we were to control an autonomous car, based on images from a front-facing camera, and we chose to use model-based RL. A model of the environment that is overall accurate could be useless for RL; while a model of the environment that is overall less accurate could be better for RL. If the more accurate model is very accurate at predicting the color of the sky, the movement of clouds or of leaves on trees, but is less accurate at predicting pedestrian movement or vehicles braking, then it is going to be less useful than a model better at doing the opposite. The core idea is that the model is not here to accurately predict the next states, but to provide meaningful information for the agent. As such, the representations of the states should reflect that objective, and the abstraction capacity of the model should be focused on the state features that are the most relevant for learning policies. This means that methods that learn a model with an objective that incorporates this policy aspect are to be preferred over methods using or learning a model that is not using an RL objective, unless that model is already very accurate and should thus already have good representations for the relevant features.

In model-based methods there can be arguments about whether or not all methods that use some form of explicit model of the environment should be qualified as model-based. Instead, we think it is better not to see the distinction as a binary choice, but more of a spectrum with various usages of a model. For example, on one extreme of the spectrum, we have pure planning methods. They are textbook model-based methods as they solely rely on the model (and not on

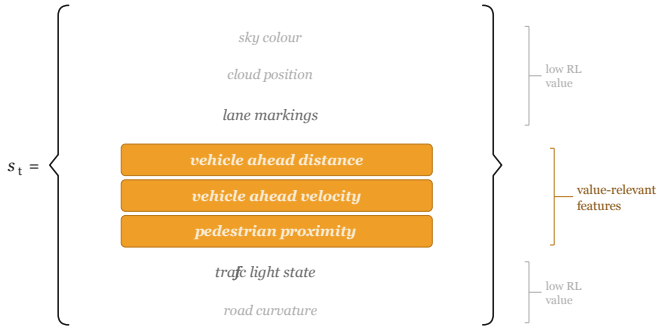



**Figure 4.2** – Example state vector for autonomous driving. Features with high RL value (amber) are those most relevant for learning a good policy, while low-value features (gray) carry little information for the agent despite being part of the observation.

experience) to derive a policy. On the opposite side of the spectrum, however, we cannot find algorithms that do not use some form of a model of the environment. For example, even for Q-learning, the table of Q-values is a model of the returns of the environment under the current policy. Nonetheless, some methods do use an explicit transition model of the environment (latent or not), which is a more commonly accepted definition of model-based. Even with this definition, the degree to which this model is used in the algorithm can vary. To clarify our position we will split this section into types of model and their usages, rather than model-free and model-based.

Pure planning methods (e.g., Monte Carlo Tree Search (MCTS, Swiechowski et al. (2021)) (Coulom, 2006; Kocsis and Szepesvári, 2006) for the games of chess and Go, or Dynamic Programming (R. Bellman, 1957; D. Bertsekas and Tsitsiklis, 1996) methods like Value Iteration (R. Bellman, 1957) and Policy Iteration (Howard, 1960)) are outside of our scope, as they do not use samples at all and only rely on the model, which is not compatible with our problem and makes no sense from a sample efficiency perspective. We are logically only interested in methods that interact with some environment to collect experience and learn from it in some way.

We will now separate model-based algorithms into three categories depending on their use of the model: Synthetic Data, Planning, and Imagination; as well as hybrid methods that cross these distinctions.

### Synthetic Data

Using synthetic data to improve sample efficiency is the first form of explicit model-based algorithm that was proposed through the Dyna (Sutton, 1991) algorithm. In this family of

algorithms, a model of the environment is used to generate synthetic experiences. The algorithm alternates between learning from real experience and synthetic experience with no distinction on the origin of the data. The goal is to "augment" the real data with a large amount of synthetic data. As for all model-based algorithms, the keystone is the model and its accuracy. Using a poor model may lead to poor policy quality, which makes model-based algorithms worse than their model-free counterparts (Janner et al., 2019). In complex applications, such models are more often than not, not available a priori. As a consequence, most modern algorithms that follow the Dyna style have opted to learn the model using the real data. The most direct example is Deep Dyna-Q (B. Peng et al., 2018) which learns a model from real data, and uses both real data and synthetic data to learn the policy. However, learning the model from real data while also learning the policy from the same real data can lead to issues.

However, Barkley and Fridovich-Keil (2025b) showed that adding synthetic data to the training process is not always beneficial. For example, while Model-Based Policy Optimization (MBPO, Janner et al. (2019)) outperforms SAC on the MuJoCo (Todorov, Erez, and Tassa, 2012) benchmark, it does not on the DeepMind Control Suite (DMC) benchmark, although they are very similar in nature. This leads to the "no free lunch" conclusion that the sample efficiency gains of such approaches may be task-dependent, and reminds us of the importance of proper benchmarking. Fortunately, Barkley and Fridovich-Keil (2025a) then investigated why this performance discrepancy between MuJoCo and DMC happened, and proposed a few corrections (independent normalization of next state and reward, and predicting next state instead of next-state residuals) to MBPO to make it outperform SAC on both benchmarks. To stay consistent with the "no free lunch" statement, these fixes may however not be sufficient for another benchmark.

### Planning

In planning-based methods, the model is not used to generate training data but is instead queried **at decision time** to search over action sequences. The agent uses the model to simulate future trajectories and selects the action leading to the highest predicted return, typically through MCTS or the Cross-Entropy Method (CEM)(R. Rubinstein, 1999; R. Y. Rubinstein and Kroese, 2004).

One of the key algorithm families in deep RL is the MCTS-based family, which includes algorithms that learn to play Go: AlphaGo (D. Silver, A. Huang, et al., 2016), AlphaGo Zero (D. Silver, Schrittwieser, et al., 2017), AlphaZero (D. Silver, Hubert, et al., 2018), MuZero (Schrittwieser et al., 2020) and finally EfficientZero (Ye et al., 2021). MuZero (Schrittwieser et al., 2020) learns a latent-space model jointly with a value function and a policy network, and uses MCTS at decision time to improve action selection. EfficientZero extends MuZero with self-supervised consistency losses, achieving stronger sample efficiency on Atari. A

key characteristic of this family is that sample efficiency comes from better decisions rather than from more gradient updates: the UTD ratio remains low (≈1–2), but each environment interaction is more informed. The main limitation is the computational cost of running tree search at every step, which can be prohibitive for real-time control tasks.

#### Imagination

The third family trains the policy *through* the model rather than alongside it. The model is used to generate imagined trajectories in a learned latent space, and the policy is then trained on these trajectories by Backpropagation Through Time (BPTT) (Werbos, 1990). This is mechanistically distinct from the Synthetic Data family, where generated transitions are used by a standard model-free update with no gradient flowing through the model, and from the Planning family, where the model is queried at decision time without gradients flowing through it.

Dreamer (Hafner, T. Lillicrap, J. Ba, et al., 2020; Hafner, T. Lillicrap, Norouzi, et al., 2021; Hafner, Pasukonis, et al., 2025) learns a Recurrent State Space Model (RSSM) in latent space and trains an actor-critic entirely within imagination, by unrolling the model for $H$ steps and backpropagating through the whole imagined trajectory. DreamerV3 (Hafner, Pasukonis, et al., 2025) extends this with improved normalization, obtaining good results across many domains with a single set of hyperparameters. IRIS (Micheli, Alonso, and Fleuret, 2023) replaces the RSSM with a discrete tokenizer and a transformer world model, showing that transformer-based world models can be highly sample-efficient on Atari. A key risk for this whole family is compounding model errors: unrolling a learned model for many steps amplifies inaccuracies and can destabilize training (Ha and Schmidhuber, 2018).

#### Hybrid Methods

Some algorithms combine two of the above mechanisms and are placed on the boundary between two columns in Figure 4.1. LOOP (Sikchi, W. Zhou, and Held, 2021) combines Synthetic Data generation with Planning, using a Dyna-style model for short rollouts while also running CEM-based planning at decision time. TD-MPC (Hansen, Su, and X. Wang, 2022) and TD-MPC2 (Hansen, Su, and X. Wang, 2024) combine Planning with Imagination/BPTT: they use CEM planning in latent space at decision time, while the latent model is trained with a temporal difference objective whose gradients flow through the learned representation. PlaNet (Hafner, T. Lillicrap, Fischer, et al., 2019) similarly uses CEM planning in the latent space of an RSSM, without an explicit actor network.

### 4.2.3 Conclusion

In practice, when dealing with real-life applications, the choice of algorithm will mainly depend on the availability of a (time-)efficient and accurate simulator. If no simulator (or at least no

efficient simulator) is available, then it is necessary to make the most out of every sample, and therefore sample-efficient algorithms are to be preferred. Otherwise, faster but less sample-efficient algorithms may be preferred, especially if they allow some form of parallelization. When it comes to deciding between model-based or model-free sample-efficient methods, the literature has not clearly given an answer. However, we believe that the decision needs to be based on the availability of a model of the system. If such a model already exists and is accurate, it will be easier to go for model-based approaches. Otherwise, avoiding learning an explicit model of the environment seems like a better solution. This favors methods like REDQ over model-based approaches, although they are not necessarily orthogonal and fusions of those algorithms may come up in the upcoming years.

## 4.3 Robust Value Estimation in Deep RL

Before diving into the literature for robust value estimation, we will first do a brief recap of value estimation in deep RL.

### 4.3.1 Value Estimation Is the Core of Deep RL

Value is a central notion in MDPs and in RL more broadly. Informally, the value of a state $V(s)$ tells how good, or bad, it is to be in a certain state $s$. Likewise, the value $Q(s, a)$ of a state-action pair $(s, a)$ tells how good it is to take action $a$ in state $s$. As knowing the optimal value function is sufficient to derive the optimal policy (D. Bertsekas and Tsitsiklis, 1996), it is clear why most RL algorithms aim to estimate value functions in one form or another.

In simple settings where the number of states and actions is finite and manageable, the optimal value can be computed through Value Iteration (R. Bellman, 1958) with full knowledge of the MDP, or learned from experience through Q-learning (Watkins and Dayan, 1992) without it. In more complex tasks, where states and actions are continuous or too numerous, value functions are approximated using neural networks. Although technically RL can be done without a value estimator (e.g., REINFORCE (R. J. Williams, 1992)), it is highly impractical and inefficient. As a consequence, modern RL algorithms all incorporate some form of value estimation, either directly (e.g., DQN (Volodymyr Mnih, Kavukcuoglu, D. Silver, Rusu, et al., 2015)) or within an actor-critic framework (e.g., PPO (Schulman, Wolski, et al., 2017), SAC (Haarnoja, A. Zhou, Abbeel, et al., 2018)).

Value networks are trained under the FQI scheme (Ernst, Geurts, and Wehenkel, 2005), which turns value estimation into a sequence of regression problems where the goal is to minimize the temporal difference (TD) error between the network predictions $f_\phi(s, a)$ and a surrogate target $\hat{Q}^\pi(s, a)$, typically a variant of the $n$-step return:

$$\text{MSE}(f_\phi, \hat{Q}^\pi) = \mathbb{E}_{(s,a)\sim\pi}[(f_\phi(s, a) - \hat{Q}^\pi(s, a))^2].$$

Since value prediction is such a core component of deep RL, the quality of this estimate directly impacts the quality of the policy. Improving the robustness of value estimation is therefore a concern that benefits a wide range of algorithms.

### 4.3.2 Policy Non-Stationarity Is Inherent to Reinforcement Learning

Value estimation in RL is a form of non-stationary online regression (Raj, Gaillard, and Saad, 2020). Unlike standard supervised learning, where inputs and targets are sampled *independent and identically distributed* (*i.i.d.*) from a fixed distribution, both the input distribution $(s, a) \sim \pi$ and the regression targets $\hat{Q}^\pi(s, a)$ change as the policy $\pi$ is updated during training. This non-stationarity is not a side-effect that can be engineered away; it is an inherent consequence of the agent learning. Igl et al. (2021) provide early evidence of this, showing that transient non-stationarities can permanently affect latent representations even in stationary environments.

It is useful to distinguish two types of non-stationarity that may arise in RL. Endogenous non-stationarity originates from within the RL process itself: it is caused solely by the policy evolving during training, while the environment remains fixed. Exogenous non-stationarity, by contrast, arises from the environment changing, for example in settings with non-stationary dynamics (Pourshamsaei and Nobakhti, 2024) or in continual learning (L. Wang et al., 2024), where the agent must solve a sequence of different tasks. In the continual learning setting, a key challenge is catastrophic forgetting (McCloskey and Cohen, 1989; Atkinson et al., 2021), where the model loses performance on previously learned tasks as new ones are introduced. This issue does not arise in the endogenous setting since we only have a single task; the value network does not need to retain prediction performance on past policies, but must remain adaptable to future ones. In this thesis we focus exclusively on endogenous non-stationarity.

A natural approach to reduce endogenous non-stationarity is to limit how much the policy changes between successive updates. TRPO (Schulman, Levine, et al., 2015) does this explicitly by constraining the KL divergence between consecutive policies, while PPO enforces a similar constraint implicitly through a clipped surrogate objective. Both approaches reduce the amount of distribution shift per update, but do not eliminate it.

### 4.3.3 Policy Non-Stationarity Consequences

The consequences of this endogenous non-stationarity have been documented under several names in the literature, each highlighting a different aspect of the same underlying problem. The prevailing consensus is that this non-stationarity leads value networks to over-emphasize the early policies encountered during training, resulting in increasing prediction errors as training progresses.

Nikishin, Schwarzer, et al. (2022) describe this as primacy bias: as the network is trained on a growing replay buffer, early transitions dominate the gradient signal. The network fits

them well, while later transitions reflecting the current policy are increasingly mispredicted. Their proposed remedy is to periodically reset parts of the agent to restore its initial learning capacity. Nikishin, Oh, et al. (2023) subsequently propose a less disruptive alternative, plasticity injection, which augments the network with freshly initialized weights without resetting existing ones. Lyle, Rowland, and Dabney (2022) describe the same phenomenon as capacity loss, emphasizing the diminishing ability of the value network to accommodate new examples as training progresses. This is particularly visible in sparse reward settings, where the network must abruptly shift its predictions when rewards are first encountered. Kumar, Agarwal, Ghosh, et al. (2021) formalize this observation under the name implicit sub-parameterization, showing that training a value estimator on sequentially non-stationary data leads to higher error rates than training on the full dataset at once, effectively reducing the expressive capacity of the network. Their proposed fix is a feature rank penalty in the objective function to prevent rank collapse in the learned representations. A complementary perspective is offered by Kumar, Agarwal, T. Ma, et al. (2022), who show that the implicit regularization induced by SGD during TD learning actively encourages feature co-adaptation, further degrading representations; they propose an explicit regularizer (DR3) to counteract this effect. Finally, Dohare et al. (2024) study plasticity loss more broadly, linking the decline in the network ability to update its weights to the emergence of dead neurons and ill-conditioned gradients, a phenomenon also observed in continual learning. Sokar et al. (2023) directly study this neuron deactivation, showing that an increasing proportion of neurons go dormant during training and proposing a recycling strategy (ReDo) to restore their expressivity.

Despite the different framings, all these works point to the same conclusion: value networks trained under policy non-stationarity tend to overfit early policies and progressively lose the ability to adapt to newer ones, leading to an accumulation of prediction error over time.

### 4.3.4 Conclusion

Value estimation is at the heart of modern deep RL, and the non-stationarity it is subjected to is not a minor perturbation but a structural property of the RL training process. The literature reviewed above shows that this has well-documented and significant consequences on the value network's capacity to follow successive policies. The existing remedies each address a specific symptom, whether through periodic resets, rank penalties, or policy constraints, but none of them target the estimation objective itself. Chapter 5 addresses this gap directly.

## 4.4 Conclusion

Three distinct bodies of literature converge in this thesis, and this chapter surveyed each of them. The first review, on RL for industrial process control, shows that RL can outperform classical controllers on setpoint-tracking problems, but that existing work tends to apply general-purpose

algorithms without tailoring them to the structural properties of industrial tasks. The second review, on sample-efficient RL, establishes that high-UTD model-free methods such as REDQ are better suited to our setting than model-based alternatives, because accurate dynamics models for complex industrial processes are rarely available. The third review, on robust value estimation, identifies a gap that directly motivates Chapter 5: while the consequences of policy non-stationarity on value networks are well-documented, no existing method addresses the estimation objective itself.

Together, these three reviews define the space of open problems this thesis sets out to fill: a method to make value estimation more robust to non-stationarity (Chapter 5), an efficient implementation platform to make large-scale experiments tractable (Chapters 6 and 7), and an algorithm that exploits the structural properties of setpoint-tracking tasks and the pre-existing experts (Chapter 8).

# Part II

# Robust Reinforcement Learning

# Chapter 5

# Robust Reinforcement Learning

> "Far better an approximate answer to the right question, which is often vague, than an exact answer to the wrong question, which can always be made precise."
>
> The Future of Data Analysis — Tukey (1962)

Modern RL algorithms frequently rely on value function approximation via neural networks trained to minimize the MSE between predictions and Bellman regression targets. However, value estimation is complicated by intrinsic policy non-stationarity during training: as the policy evolves, both the data distribution and the regression targets shift, progressively degrading the network's capacity to adapt. To address this, we introduce EVarEst, a novel value network objective that generalizes the MSE by re-weighting its bias-variance decomposition via a single scalar hyperparameter $\alpha_{\text{ev}}$. Setting $\alpha_{\text{ev}} = 1/2$ recovers the MSE; lower values increasingly penalize the Variance of Residual Errors (VRE), while the limit $\alpha_{\text{ev}} \to 0$ approaches the Actor with Variance Estimated Critic (AVEC) objective. EVarEst can replace the value network loss in any RL algorithm with minimal code modification. Empirically, EVarEst with task-appropriate $\alpha_{\text{ev}}$ outperforms the MSE baseline on the majority of MuJoCo and AdroitHand tasks for both SAC (average improvement: 13.29%) and PPO (28.7%), and systematically avoids the catastrophic failures exhibited by AVEC. Our analysis shows that lower $\alpha_{\text{ev}}$ values maintain higher feature rank throughout training, preserving network plasticity and the capacity to re-fit successive policy distributions. However, our initial hypothesis that variance penalization would produce more stable value losses across policy updates is not supported: lower $\alpha_{\text{ev}}$ in fact increases loss variation, which we interpret as a signature of a more responsive, adaptive network rather than harmful instability. The principal limitation of EVarEst is the sensitivity of performance to $\alpha_{\text{ev}}$, which varies widely across tasks and currently requires grid search to set.

## Contents

5.1 Introduction . . . . . 65
5.2 Related works . . . . . 66
5.3 Background and notations . . . . . 67
5.4 EVarEst . . . . . 69
5.5 Experimental study . . . . . 72
5.6 Conclusion . . . . . 80

## 5.1 Introduction

As established in Chapter 1, modern deep RL algorithms represent the value function $Q^\pi$ (or $V^\pi$) using a neural network trained under the FQI scheme, minimizing the MSE between predictions and regression targets derived from the Bellman equation. This turns value estimation into a sequence of supervised regression problems. However, this sequential setting is fundamentally different from classical supervised learning because the regression target is **non-stationary**: as the agent learns, the policy $\pi$ evolves, shifting both the data distribution $(s, a) \sim \pi$ and the values to predict.

**Consequences of non-stationarity.** This non-stationarity introduces the need for the neural network to adapt to new sample distributions over time. In practice, value networks lack robustness to this perturbation (Lyle, Rowland, and Dabney (2022), Kumar, Agarwal, Ghosh, et al. (2021), Nikishin, Schwarzer, et al. (2022), Dohare et al. (2024)), they may fit the earlier examples correctly but fail to correctly predict newer ones. Since value prediction is such a core component of Deep RL, improving the robustness to this intrinsic non-stationarity would benefit a wide range of algorithms.

Standard Stochastic Gradient Descent (SGD) training assumes i.i.d. data, an assumption that is violated here: successive transitions are correlated and drawn from a distribution that shifts with the policy. This further complicates stable optimization, as discussed in Section 4.3.

**Bias-variance weighting in RL as an answer to the problem of non-stationarity.** In this work, we design a value prediction objective better suited to mitigate the aforementioned incapacity to adapt to new policies. The standard value objective, the MSE can be expressed as a combination of a bias and a variance (Geman, Bienenstock, and Doursat, 1992). In this decomposition, we introduce a new hyperparameter $\alpha_{\text{ev}}$, and we propose to weigh the two terms with $\alpha_{\text{ev}}$ and $1 - \alpha_{\text{ev}}$ respectively. The resulting objective reduces back to MSE for $\alpha_{\text{ev}} = 1/2$. We name this objective EVarEst. The idea is that in some situations, it is better to emphasize VRE over bias to promote robustness against policy non-stationarity. In Section 5.4.1, we motivate this weighting between bias and variance. In Section 5.4.2, we formalize EVarEst as a penalization of the MSE and introduce its connection to AVEC (Flet-Berliac et al., 2021), a related objective that only considers the VRE and has shown promising performance on MuJoCo (Flet-Berliac et al., 2021) and soft-robotics (Schegg et al., 2023). Both MSE and AVEC are special cases of EVarEst. In Section 5.5, we compare EVarEst to both baselines experimentally and investigate whether it delivers on its theoretical motivations.

**Contributions.** In this chapter, we make the following set of contributions:

- We motivate and introduce EVarEst, a new objective function enabling the control of the relative weight of bias and VRE using a new hyperparameter $\alpha_{\text{ev}}$. We emphasize that

EVarEst (1) requires minimal modifications in the code and (2) is compatible with any algorithm using a variation of the MSE as it only requires changing the value network objective function, while not introducing any term of new nature.

- We provide experimental results on the MuJoCo and AdroitHand continuous control benchmark, testing a wide variety of $\alpha_{\text{ev}}$ values for EVarEst coupled to SAC and PPO. We experimentally demonstrate that EVarEst coupled to SAC with the proper $\alpha_{\text{ev}}$ value leads to significant improvements over using the standard MSE value objective or AVEC coupled to SAC.
- We analyze the performance of EVarEst, develop intuitions about how it impacts value estimation, and experimentally challenge those intuitions.

A preliminary version of this work was presented as a poster at the European Workshop on Reinforcement Learning (EWRL) 2025.

Before delving into EVarEst, in the next two sections we will (1) go over the related works and (2) introduce the necessary background.

## 5.2 Related works

We have already covered the literature regarding non-stationarity and its impact on value learning in deep RL in Section 4.3. However, we have not discussed the genesis of EVarEst, and methods similar to EVarEst.

**AVEC** EVarEst grew directly out of an attempt to understand and generalize AVEC, proposed by former members of our team (Flet-Berliac et al., 2021). AVEC drops the bias term from the value objective entirely, keeping only the VRE, and reports consistent performance gains across MuJoCo at virtually no added implementation cost. Its simplicity made it a compelling foundation to investigate further. However, the original paper does not provide a formal account of why removing the bias term entirely, rather than merely reducing it, should be the right choice under non-stationarity. This raised a natural question: is AVEC's all-or-nothing treatment of variance optimal, or is there an intermediate trade-off between bias and VRE that performs better across a wider range of tasks? Starting from the bias-variance decomposition of the MSE, we formalized what AVEC was doing implicitly and generalized it to a continuous family of objectives parameterized by a single scalar $\alpha_{\text{ev}}$. This line of inquiry led directly to EVarEst.

**Similar methods** EVarEst is a penalization of the MSE using a data-dependent penalization term (instead of a term solely depending on the model itself like in regularization). It is also equivalent to a reweighting of the bias and VRE terms of the MSE decomposition. Similar

methods have already been explored in supervised classification, like reweighting the asymmetry in ROC (Bach, Heckerman, and Horvitz, 2006), or composite losses (Reid and Williamson, 2009). However in regression this is a less frequent method. After identifying the penalization formulation of EVarEst, only Sample Variance Penalization (SVP) (Maurer and Pontil, 2009) comes close to ours. SVP penalizes using the square-root of the variance of the squared residual errors.

Although this method is similar to our proposed solution, there are some noteworthy differences: (1) SVP is aimed at improving generalization and not mitigating non-stationarity related issues. (2) The use of the squared residual errors instead of the residual errors makes underestimation and overestimation indistinguishable, e.g., if half the samples satisfy $\hat{f}(x) = f(x) + 1$ and the other half satisfy $\hat{f}(x) = f(x) - 1$, all squared residuals equal 1 and their variance is zero, so the SVP penalization vanishes, while these errors remain penalized in EVarEst. For these reasons, although SVP is well studied and comes with theoretical guarantees, we still think EVarEst is better fitted for the RL setting.

Apart from AVEC, which can be viewed as an extreme re-weighting of the MSE, no prior work has studied the impact of bias vs VRE weighting for the value prediction objective in RL.

## 5.3 Background and notations

**Notation conventions for this chapter.** Throughout this chapter, $\alpha_{\text{ev}} \in (0, 1]$ denotes the EVarEst bias-variance weighting parameter introduced below. This is distinct from the SAC entropy temperature $\alpha_{\text{ev}}$ used in Chapter 1; both symbols refer to scalars that trade off two objectives, but they appear in entirely separate contexts and equations. The value (or critic) network is written as $f_\phi$ to make the generic, algorithm-agnostic nature of the derivations explicit; it corresponds directly to $Q_\phi(s, a)$ (or $V_\phi(s)$) as introduced in Chapter 1, with $\phi$ denoting the network weights.

### 5.3.1 Value estimation in Deep Reinforcement Learning

Actor-critic algorithms (or pure value algorithms like DQN) use a **value (or critic) network**, denoted $f_\phi$ (corresponding to $Q_\phi$ or $V_\phi$, see Chapter 1), to represent the value function $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ (or $V^\pi$) under policy $\pi$. Learning to approximate the value function is done by training the network to minimize an objective function $J$, most commonly the mean squared error (MSE) between the predictions $f_\phi(s, a)$ and $\hat{Q}^\pi(s, a)$, a surrogate for $Q^\pi(s, a)$:

$$\text{MSE}(f_\phi, \hat{Q}^\pi) = E_{(s,a)\sim\pi}[(f_\phi(s, a) - \hat{Q}^\pi(s, a))^2]. \tag{5.1}$$

The value surrogate $\hat{Q}^\pi$ is usually a variation of the $n$-step return (see Sutton and A. Barto (2020, chapter 7.1 p. 143)):

$$\hat{Q}^\pi(s_t, a_t) = \sum_{k=0}^{n-1} \gamma^k r_{t+k+1} + \gamma^n \hat{V}^\pi(s_{t+n}).$$

It is important to note that $\hat{Q}^\pi$ is not the ground-truth value function $Q^\pi$, but a bootstrap estimate: the target is computed using the same value network whose parameters are being updated, typically via a lagged copy (target network). Consequently, the regression target is itself an approximation subject to its own bias and variance. Errors in the current value network therefore propagate into subsequent targets, a compounding effect that is well-documented in the deep RL literature (Sutton and A. Barto, 2020) and that makes value estimation fundamentally harder than standard supervised regression with fixed labels. The analysis in this chapter focuses on the prediction error relative to the surrogate $\hat{Q}^\pi$ rather than the true $Q^\pi$, since the latter is not accessible during training.

### 5.3.2 Non-stationarity in Value estimation in Deep RL

Value prediction is a sequence of regression problems; however, it differs from classic supervised learning because of the non-stationarity introduced by policy changes during training. As the agent learns, the policy $\pi$ is updated. These changes in $\pi$ modify the distributions of (1) targets $\hat{Q}^\pi$ and (2) samples $(s, a) \sim \pi$, which makes the objective $J$ (see eq. (5.1)) non-stationary during training. Instead of wanting our model $f_\phi$ to minimize the objective[1] $J$, we are instead trying to minimize the expected cumulative error (Raj, Gaillard, and Saad, 2020) over time:

$$J_n = \sum_{i=0}^{n} \mathbb{E}_{(s,a)\sim\pi_i} \left[ (f_\phi(s, a) - \hat{Q}^{\pi_i})^2 \right],$$

with $n$ the current number of policy updates. In order to train a model to minimize $J_n$ over time, this model needs to be able to adapt to policy updates. In practice, the opposite effect is observed and the value model at a step $t$ tends to overfit the policy at that step. This issue is known under various names in the literature: capacity loss (Lyle, Rowland, and Dabney, 2022), primacy bias (Nikishin, Schwarzer, et al., 2022), or plasticity loss (Dohare et al., 2024). Overall the underlying trouble is the same: the model tends to fit the earlier samples but loses the capacity to adapt to newer samples over time, which leads to larger and larger cumulative error over time.

The capacity of a model to adapt to a new data distribution is called robustness. A robust model has a small loss change when subject to small variations in distribution. To formalize

[1] We use the term objective $J$ instead of the risk $R$ to avoid confusion with the reward function $R$. As we are minimizing the objective, both terms are equivalent here.

this characteristic, we refer to the early work on robustness in Hampel (1971). In the context of learning with data sampled from a non-stationary distribution $P_t$ that can change from one iteration $t$ to the next, if some distance between distributions $d(P_t, P_{t+1})$ is small then an estimator is robust to this distance if its value on $P_t$ is not far from its value on $P_{t+1}$, this is the type of robustness we target in this article.

## 5.4 EVarEst

### 5.4.1 Motivation

Value prediction is a core component of RL, and this is why mitigating the issues that arise from its non-stationary setting is crucial. We would like to improve the capacity of the value network to adapt to successive distributions from successive policies in order to get consistent and low prediction errors along the training. Our idea to mitigate these issues stems from observing the most common objective for value prediction, the MSE. The MSE can be decomposed in terms of bias and VRE (Geman, Bienenstock, and Doursat, 1992):

**Lemma 5.1.**

*For any estimator $\hat{f}$ and its target $f$, we have:*

$$MSE(\hat{f}, f) = Bias(\hat{f}, f)^2 + Var(\hat{f} - f) \tag{5.2}$$

*where:*

$$Bias(\hat{f}, f) = \mathbb{E}_x[\hat{f}(x) - f(x)]$$
$$Var(\hat{f} - f) = \mathbb{E}_x\left[(\hat{f}(x) - f(x) - \mathbb{E}_x[\hat{f}(x) - f(x)])^2\right].$$

**Proof.** See Appendix A.1.1. □

In this decomposition, the bias and the variance are equally weighted. In order to adapt to future policies, we argue that we should focus on the minimization of the VRE rather than on the minimization of the bias. The intuition is that by focusing on minimizing the VRE, we incentivize the model to learn to reduce errors in a consistent way over all samples, rather than focusing on predicting well a subset of samples, and badly predicting the others. We hypothesize that this way, the network will learn features that are more "generalizable"[2] across

[2]This is not the usual notion of generalization as we consider the performance across different distributions instead of different samples from the same distributions.

policies. In addition, other works (Ilyas et al. (2020),Flet-Berliac et al. (2021)) suggest that variance is a bigger problem than bias when it comes to value prediction.

Moreover, the MSE is a theoretically consistent objective: its bias vanishes asymptotically as the number of training samples grows. However, in the non-stationary RL setting, each policy update resets the regression problem before convergence is reached. The asymptotic regime is therefore never attained, and the bias-variance trade-off is better evaluated in the finite-sample regime relevant to each individual policy update. In this regime, accepting a non-zero bias in exchange for lower VRE can improve the consistency of prediction errors across the sample distribution and across successive updates, with the intended effect of reducing cumulative value estimation error over training.

As a consequence, we argue that the value estimation objective should reflect this non-stationary setting by promoting two properties: (1) low VRE, even at the cost of some transient bias (so that prediction errors are consistent across samples and between policy updates), and (2) maintained network plasticity (preserving the capacity to re-fit the value function when the policy changes, rather than overfitting to earlier distributions). If criterion (2) is not met, the model undergoes capacity loss: it fits the first policies well but progressively loses the ability to adapt to new ones. If criterion (1) is not met, the network may retain plasticity in principle but fail to minimize the objective consistently enough before the distribution shifts again. We hypothesize that both goals could be achieved through a deliberate re-weighting of bias and VRE, and that better value prediction objectives than the MSE exist taking exactly this form.

### 5.4.2 Method

In order to study the weighting of bias and VRE as alternative value network objectives, we introduce a weight $\alpha_{\text{ev}}$ into the MSE bias-variance decomposition (eq. (5.2)):

$$\mathcal{L}(\alpha_{\text{ev}}) \triangleq \alpha_{\text{ev}} Bias(\hat{f}, f)^2 + (1 - \alpha_{\text{ev}}) Var(\hat{f} - f) \tag{5.3}$$

with $\alpha_{\text{ev}} \in [0, 1]$ the bias-variance weighting parameter. Note that $\alpha_{\text{ev}} = 0$ gives $\mathcal{L}(0) = \text{Var}(\hat{f} - f)$, recovering the AVEC objective exactly. The EVarEst definition below divides by $\alpha_{\text{ev}}$ and therefore requires $\alpha_{\text{ev}} \in (0, 1]$; AVEC is recovered in the limit $\alpha_{\text{ev}} \to 0$.

This objective can also be expressed as a penalized MSE. Dividing $\mathcal{L}(\alpha_{\text{ev}})$ by $\alpha_{\text{ev}} > 0$ does not change the minimizer (scale-invariant optimizers such as Adam are unaffected), but makes the MSE term appear explicitly and frames EVarEst as a familiar baseline augmented with a single data-dependent penalty term:

**Definition 5.1** Error-Variance penalized Estimation (EVarEst)

For any estimator $\hat{f}$, its target $f$, and a bias-variance weighting parameter $\alpha_{\text{ev}} \in (0, 1]$, the **Error-Variance penalized Estimation** objective is defined as:

$$\mathcal{L}_{EVarEst}(\alpha_{\text{ev}}) \triangleq \frac{\mathcal{L}(\alpha_{\text{ev}})}{\alpha_{\text{ev}}} = \underbrace{\text{MSE}(\hat{f}, f)}_{\text{standard objective}} + \underbrace{\frac{1 - 2\alpha_{\text{ev}}}{\alpha_{\text{ev}}} \text{Var}(\hat{f} - f)}_{\text{variance penalty}}. \tag{5.4}$$

For $\alpha_{\text{ev}} < 1/2$ the penalty coefficient $(1 - 2\alpha_{\text{ev}})/\alpha_{\text{ev}}$ is positive, incentivizing lower VRE. At $\alpha_{\text{ev}} = 1/2$ the penalty vanishes and EVarEst reduces to the MSE. For $\alpha_{\text{ev}} > 1/2$ the coefficient is negative, reducing the weight placed on variance relative to bias. Setting $\alpha_{\text{ev}} \to 0$ approaches the AVEC objective (Flet-Berliac et al., 2021), which discards the bias term entirely.

The formulation in equation (5.4) makes explicit the difference between a principled re-weighting of the bias-variance decomposition of the MSE and an objective that discards the bias entirely, as done in AVEC (Flet-Berliac et al., 2021). Because AVEC is not a penalization of the MSE, it does not enjoy the theoretical guarantees of the MSE. The same remark holds for $\alpha_{\text{ev}} = 1$, which only considers the bias.

We emphasize that unlike regularization or auxiliary objectives, EVarEst does not introduce any term of new nature: it only re-weights the bias and variance already present in the MSE decomposition.

Applied to value estimation in RL (replacing $\hat{f}$ with the value network $f_\phi$ and $f$ with the surrogate $\hat{Q}^\pi$), the objective becomes:

$$\mathcal{L}_{EVarEst}(\alpha_{\text{ev}}) \triangleq \text{MSE}(f_\phi, \hat{Q}^\pi) + \frac{1 - 2\alpha_{\text{ev}}}{\alpha_{\text{ev}}} \text{Var}(f_\phi - \hat{Q}^\pi).$$

This objective can replace the value network loss in any RL algorithm that uses a variant of the MSE.

**Remark.** For $\alpha_{\text{ev}} \leq 1/2$, unlike other penalizations (SVP, $\ell_1$ regularization, *etc.*), our objective function is upper and lower bounded by a constant multiple of the MSE:

$$\text{MSE}(f_\phi, \hat{Q}^\pi) \;\leq\; \mathcal{L}_{EVarEst}(\alpha_{\text{ev}}) \;\leq\; \left(1 + \frac{1 - 2\alpha_{\text{ev}}}{\alpha_{\text{ev}}}\right) \text{MSE}(f_\phi, \hat{Q}^\pi).$$

In particular, for $\alpha_{\text{ev}}$ close to $1/2$, the optimization done by EVarEst remains close to minimizing the MSE. Conversely, for small $\alpha_{\text{ev}}$ the upper bound grows as $(1 - \alpha_{\text{ev}})/\alpha_{\text{ev}} \cdot \text{MSE}$, becoming uninformative in the regime where EVarEst most departs from the MSE baseline.

## 5.5 Experimental study

To measure EVarEst performance, we embed it into two popular RL algorithms, SAC and PPO, creating EVarEst-SAC and EVarEst-PPO. SAC and PPO are described in full in Chapter 1. For the purposes of this evaluation, the key distinction is that SAC derives policy updates directly from the critic, making value estimation error feed immediately into the policy gradient, while PPO uses the advantage estimated via GAE ($\lambda = 0.95$), which blends actual returns and value predictions and makes the critic's influence more indirect. We chose SAC and PPO to cover both off-policy and on-policy regimes and to demonstrate the broad applicability of EVarEst.

For evaluation, we use the MuJoCo (Todorov, Erez, and Tassa, 2012) and AdroitHand (Rajeswaran et al., 2018) benchmarks, covering locomotion and dexterous manipulation tasks respectively. All algorithms performance were evaluated over 10 seeds. Experiments were made using Stable Baselines3 (SB3, Raffin et al. (2021)) implementations of SAC and PPO and applying modifications for the EVarEst loss and the logging of the metrics reported in the chapter. For more details about the experimental setup, see Appendix A.4. In order to compare the performance of the different values of $\alpha_{\text{ev}}$, we follow the methodology from Agarwal et al. (2021) which promotes the use of mean, median and inter-quartile mean (IQM), as well as confidence intervals.

### 5.5.1 Benchmarks

**MuJoCo** MuJoCo (Multi-Joint dynamics with Contact, Todorov, Erez, and Tassa 2012) is a physics simulator for articulated rigid bodies, providing the standard continuous control benchmark suite in deep RL via the Gymnasium interface (Towers et al., 2024). All tasks are locomotion tasks: a simulated creature learns to move forward using proprioceptive observations and a dense reward based on forward velocity and energy efficiency. We evaluate on five tasks of varying complexity: **HalfCheetah**, **Hopper**, **Walker2d**, **Ant**, and **Humanoid**, illustrated in Figure 5.1.

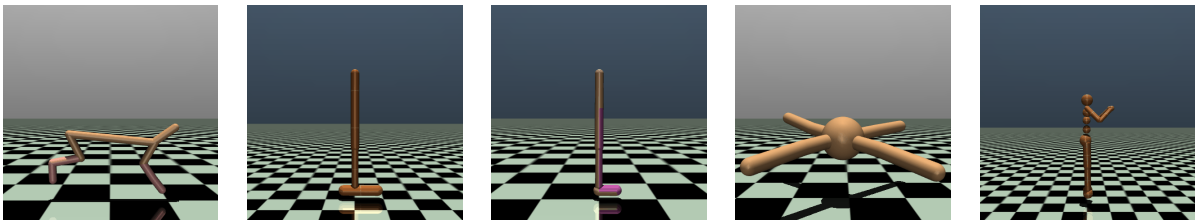


**Figure 5.1** – MuJoCo locomotion tasks (left to right): HalfCheetah, Hopper, Walker2d, Ant, Humanoid.

**AdroitHand** AdroitHand (Rajeswaran et al., 2018) is a benchmark suite for dexterous manipulation using a 24-degree-of-freedom simulated robotic hand. Unlike the MuJoCo locomotion

tasks, rewards are sparse or structured and tasks require fine-grained finger control. We evaluate on four tasks: **Door**, **Hammer**, **Pen**, and **Relocate**, illustrated in Figure 5.2.

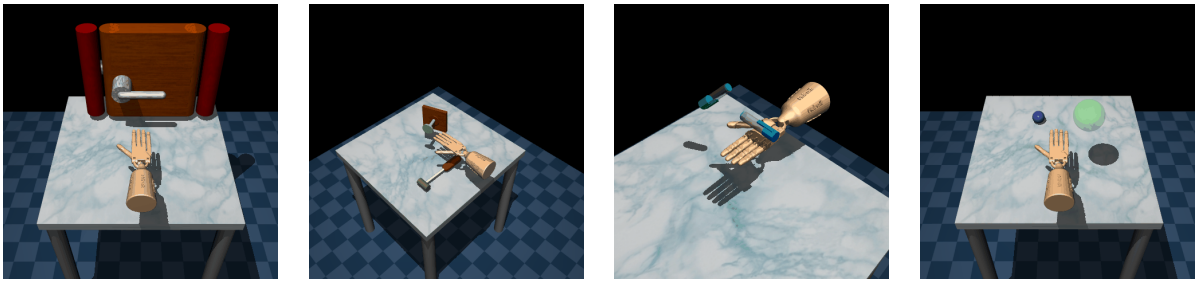
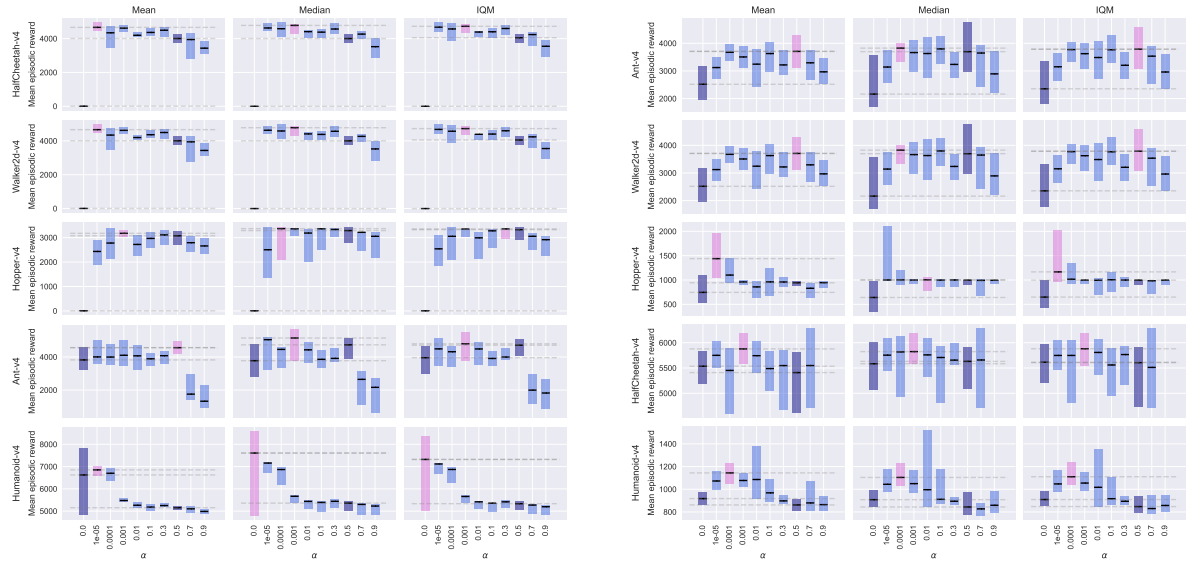


**Figure 5.2** – AdroitHand dexterous manipulation tasks (left to right): Door, Hammer, Pen, Relocate.

### 5.5.2 EVarEst policy performance

We study the impact of the weighting parameter $\alpha_{\text{ev}}$ on policy performance, comparing EVarEst-SAC and EVarEst-PPO against their respective baselines across the MuJoCo and AdroitHand benchmark suites.

#### Continuous control on MuJoCo




(**a**) EVarEst-SAC

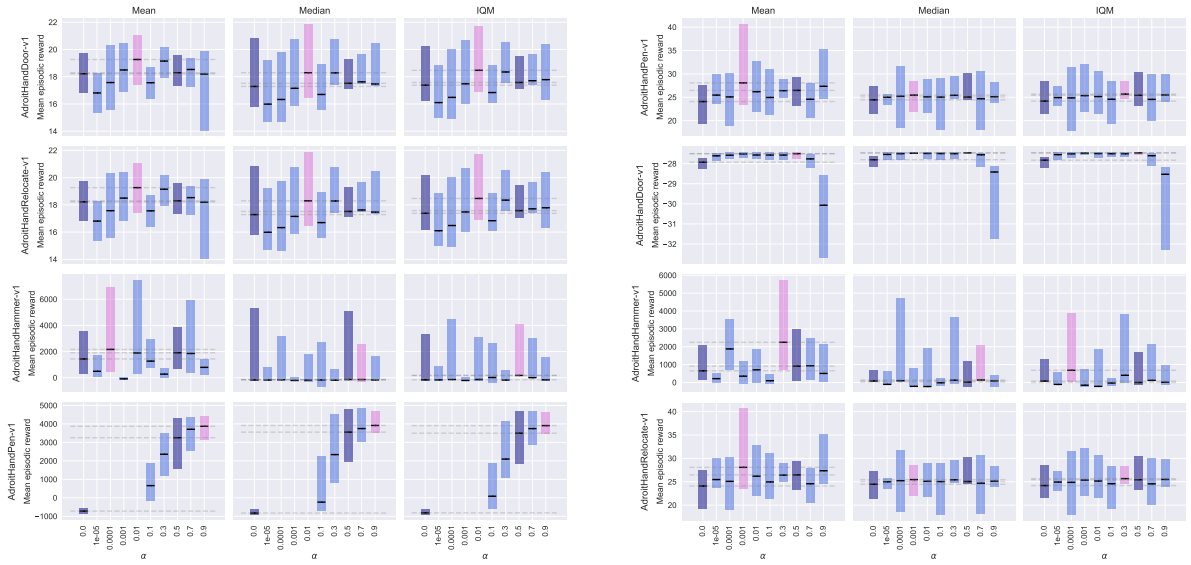
(**b**) EVarEst-PPO

**Figure 5.3** – Average performance over the last $10^5$ training timesteps for EVarEst-SAC (left) and EVarEst-PPO (right) on MuJoCo with different $\alpha_{\text{ev}}$ values. Y-axis: Average episodic returns. Black lines represent the metric specified at the top of the column over 10 seeds. Bars represent 95% bootstrap confidence intervals. Pink is the best performing $\alpha_{\text{ev}}$, dark blue are the baselines.

In Figure 5.3, we observe that, for SAC, $\alpha_{\text{ev}}$ close to 0 leads to equal or better performance than larger values, notably the MSE ($\alpha_{\text{ev}} = 0.5$), except for Ant-v4 where the differences are limited.

For PPO, the relation is more dependent on the environment, as smaller values (Hopper, Humanoid) and larger values (Walker2d) can both lead to the best results. Interestingly, the behavior can vary dramatically when using AVEC (equivalent to EVarEst with $\alpha_{\text{ev}} = 0$): we observe that on two tasks, Hopper and Walker2d, it instead leads to the worst performance, seemingly not learning at all for SAC.[3]

### Continuous control on AdroitHand




(**a**) EVarEst-SAC (**b**) EVarEst-PPO

**Figure 5.4** – Average performance over the last $10^5$ training timesteps for EVarEst-SAC (left) and EVarEst-PPO (right) on AdroitHand with different $\alpha_{\text{ev}}$ values. Y-axis: Average episodic returns. Black lines represent the metric specified at the top of the column over 10 seeds. Bars represent 95% bootstrap confidence intervals. Pink is the best performing $\alpha_{\text{ev}}$, dark blue are the baselines.

In Figure 5.4 we can observe how AdroitHand tasks differ from MuJoCo tasks, as a smaller $\alpha_{\text{ev}}$ is no longer correlated with better performance. Most interestingly, on AdroitHandPen the relation seems to be the opposite with a higher bias leading better results. In this task, the robot hand needs to manipulate a pen to match a given orientation and location. However the pen can be dropped leading to important negative rewards until the end of the fixed length episode. We slightly modified the task so that when the robot drops the pen, the episode terminates. With this modification of the task, the anomaly is reduced (see Appendix A.2.2). This illustrates that task design has significant impact on results, and that allowing for early termination in this case is beneficial. Indeed, if the task is not stopped, then the agent observes bad returns which are essentially meaningless for the task since the robot dropped the pen, so that any action of the robot is simply useless. Actually, this is a key remark concerning the

[3] In the original AVEC paper (Flet-Berliac et al., 2021) reports stable performance on Walker2d (for SAC) and HalfCheetah (for PPO); we were unable to reproduce these results. Hopper is not considered in their work.

design of the task: there is no hope to learn anything if the return signal does not contain any useful information.

### 5.5.3 Overall improvements

| Environment | SAC ($\alpha_{\text{ev}} = 0.5$) | AVEC-SAC ($\alpha_{\text{ev}} = 0$) | EVarEst-SAC ($\alpha^*_{\text{ev}}$) |
|---|---|---|---|
| Ant-v4 | **4563.3 ± 1783.8** | 3821.9 ± 851.4 (-16.2%) | 4102.6 ± 2216.7 (-10.1%), $\alpha_{\text{ev}} = 10^{-3}$ |
| HalfCheetah-v4 | 10730.1 ± 799.8 | 11346.6 ± 464.3 (5.7%) | **11405.9 ± 536.1 (6.3%)**, $\alpha_{\text{ev}} = 10^{-5}$ |
| Hopper-v4 | 3074.5 ± 11.6 | 4.4 ± 655.2 (-99.9%) | **3175.3 ± 504.3 (3.3%)**, $\alpha_{\text{ev}} = 10^{-3}$ |
| Humanoid-v4 | 5154.2 ± 2874.4 | 6624.5 ± 790.4 (28.5%) | **6852.2 ± 1127.8 (32.9%)**, $\alpha_{\text{ev}} = 10^{-5}$ |
| Walker2d-v4 | 4006.5 ± 68.4 | 5.8 ± 663.4 (-99.9%) | **4664.2 ± 773.4 (16.4%)**, $\alpha_{\text{ev}} = 10^{-5}$ |
| AdroitHandDoor-v1 | ≈**18 ± ?**† | ≈1.5 ± ?† (-91.9%) | ≈13.5 ± ?† (-25.0%), $\alpha_{\text{ev}} = 0.1$ |
| AdroitHandHammer-v1 | 1910.7 ± 3948.8 | 1435.0 ± 3802.5 (-24.9%) | **2164.9 ± 5013.0 (13.3%)**, $\alpha_{\text{ev}} = 10^{-4}$ |
| AdroitHandPen-v1 | 3255.3 ± 348.6 | -719.5 ± 2517.1 (-122.1%) | **3870.8 ± 1975.2 (18.9%)**, $\alpha_{\text{ev}} = 0.9$ |
| AdroitHandRelocate-v1 | 18.3 ± 6.1 | 18.2 ±6.7 (-0.5%) | **19.3 ± 6.7 (5.5%)**, $\alpha_{\text{ev}} = 0.01$ |
| **Average Improvement** | | 5.43% | **13.29%** |

**Table 5.1** – Average performance (± std) over the last $10^5$ timesteps. Comparison of SAC, AVEC-SAC, and EVarEst-SAC ($\alpha^*_{\text{ev}}$: the best performing $\alpha_{\text{ev}}$). † A logging error caused the loss of raw episode returns for the AdroitHandDoor-v1 runs; only relative performance percentages were retained. The reported averages are reconstructed from the baseline score and the relative performance, so no standard deviation is available.

| Environment | PPO ($\alpha_{\text{ev}} = 0.5$) | AVEC-PPO ($\alpha_{\text{ev}} = 0$) | EVarEst-PPO ($\alpha^*_{\text{ev}}$) |
|---|---|---|---|
| Ant-v4 | 2238.0 ± 827.2 | 2381.9 ± 540.7 (6.4%) | **2486.2 ± 722.6 (11.1%)**, $\alpha_{\text{ev}} = 0.9$ |
| HalfCheetah-v4 | 5409.4 ± 620.0 | 5534.9 ± 852.7 (2.3%) | **5877.7 ± 575.8 (8.7%)**, $\alpha_{\text{ev}} = 10^{-3}$ |
| Hopper-v4 | 943.2 ± 562.2 | 745.5 ± 155.6 (-21.0%) | **1439.3 ± 786.3 (52.6%)**, $\alpha_{\text{ev}} = 10^{-5}$ |
| Humanoid-v4 | 862.0 ± 173.0 | 917.8 ± 154.0 (6.5%) | **1144.1 ± 285.1 (32.7%)**, $\alpha_{\text{ev}} = 10^{-4}$ |
| Walker2d-v4 | **3706.7 ± 1097.5** | 2516.9 ± 1147.8 (-32.1%) | 3673.8 ± 673.4 (-0.9%), $\alpha_{\text{ev}} = 10^{-4}$ |
| AdroitHandDoor-v1 | **-27.5 ± 0.4** | -27.9 ± 0.3 (-1.5%) | -27.5 ± 0.3 (0.0%), $\alpha_{\text{ev}} = 10^{-3}$ |
| AdroitHandPen-v1 | 1449.7 ± 924.9 | 1056.2 ± 1119.3 (-27.1%) | **1485.8 ± 1093.9 (2.5%)**, $\alpha_{\text{ev}} = 0.7$ |
| AdroitHandHammer-v1 | 911.1 ± 1768.6 | 653.0 ± 2376.7 (-28.3%) | **2246.8 ± 4065.6 (146.6%)**, $\alpha_{\text{ev}} = 0.3$ |
| AdroitHandRelocate-v1 | 26.5 ± 8.9 | 24.1 ± 7.1 (-9.1%) | **28.1 ± 13.0 (6.0%)**, $\alpha_{\text{ev}} = 0.001$ |
| **Average Improvement** | | -11.65% | **28.7%** |

**Table 5.2** – Average performance (± std) over the last $10^5$ timesteps. Comparison of PPO, AVEC-PPO, and EVarEst-PPO ($\alpha^*_{\text{ev}}$: the best performing $\alpha_{\text{ev}}$)

Tables 5.1 and 5.2 summarize the performance of EVarEst-SAC and EVarEst-PPO with the best-found $\alpha^*_{\text{ev}}$ per task, alongside the MSE and AVEC baselines. Several observations stand out.

First, EVarEst with tuned $\alpha^*_{\text{ev}}$ outperforms the MSE baseline on the majority of tasks for both algorithms (8 out of 9 for SAC, 7 of 9 for PPO), with average improvements of 13.29% and 28.7% respectively, confirming that the MSE's implicit equal weighting of bias and variance is not universally optimal.

Second, AVEC ($\alpha_{\text{ev}} = 0$) is unreliable: it catastrophically fails on Walker2d and Hopper for SAC ($-99.9\%$) and delivers a negative average improvement of $-11.65\%$ for PPO. EVarEst

dominates AVEC precisely because it allows intermediate values of $\alpha_{\text{ev}}$ that avoid these failure modes while still benefiting from some VRE reduction.

Third, the optimal $\alpha^*_{\text{ev}}$ varies widely across tasks, from $10^{-5}$ to $0.9$ for SAC, with no single value performing best everywhere. This task-dependence is the principal limitation of EVarEst in its current form and motivates the analysis in Section 5.5.4.

### 5.5.4 EVarEst impact on learning

Having observed that EVarEst generally outperforms both the MSE and AVEC objectives when tuned with the appropriate $\alpha_{\text{ev}}$ value, we now turn to an experimental investigation of how EVarEst affects the learning process. Specifically, we aim to test the two criteria from Section 5.4.1: (1) whether lower $\alpha_{\text{ev}}$ promotes maintained network plasticity (measured via feature rank, i.e., the number of linearly independent directions in the last hidden layer's activations, which reflects how much of the network's representational capacity is actively used), and (2) whether it also improves stability of the value loss across successive policy updates, which we use as a proxy for robustness to non-stationarity. We also evaluate the direct impact of $\alpha_{\text{ev}}$ on value estimation error.

#### Adaptability to non-stationarity

In order to quantify how much EVarEst impacts the adaptability of the estimator with regards to new samples from successive policies we measure the feature rank (Lyle, Rowland, and Dabney, 2022) of the model throughout training. The feature rank is computed as the number of singular values of the feature matrix that are greater than a given threshold $\varepsilon$. For SAC this threshold is kept at $\varepsilon = 0.01$ as in Lyle, Rowland, and Dabney (2022), for PPO however we increased it to $\varepsilon = 0.1$ as the initial value did not allow us to observe any difference between $\alpha_{\text{ev}}$ values. We focus on SAC here for conciseness; results for PPO can be found in Appendix A.2.4.

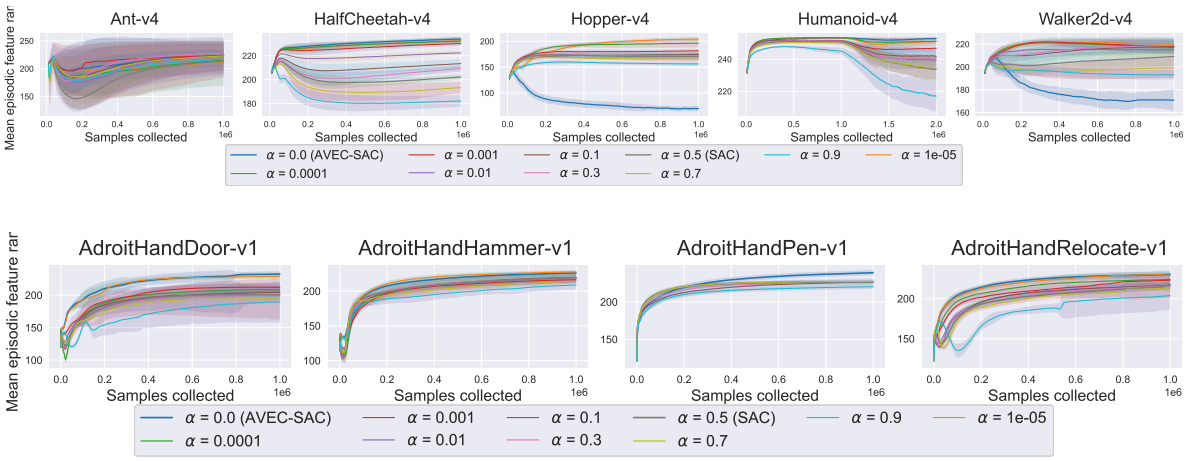



**Figure 5.5** – Feature rank of EVarEst-SAC evolution during training (Top: Mujoco, Bottom: AdroitHand)

In Figure 5.5 we can note that lower values of $\alpha_{\text{ev}}$ generally promote higher feature ranks on MuJoCo tasks, with the notable exception of Walker2d and Hopper where AVEC ($\alpha_{\text{ev}} = 0$) collapses: feature rank drops sharply and performance fails entirely. The strong co-variation of feature rank and performance across tasks supports the hypothesis that lower $\alpha_{\text{ev}}$ maintains network plasticity, preserving the capacity to adapt to policy shifts. This is criterion (1) from Section 5.4.1, and the evidence here broadly supports it on locomotion tasks with reliable reward signals.

The AdroitHand results reveal a qualitatively different regime. On AdroitHandPen, AVEC achieves the *highest* feature rank but the *worst* performance (see also Appendix A.2.2). This apparent contradiction is explained by the structure of the task: when the robot drops the pen, the episode continues and the agent receives a long sequence of highly negative, essentially uninformative returns. A high-rank network faithfully fits this complex but meaningless signal, building a value function that reflects noise rather than task-relevant structure. A more biased, lower-rank estimator is less responsive to this noise, producing a coarser but more useful value signal for policy updates. This suggests that the benefit of lower $\alpha_{\text{ev}}$ (high adaptability, high rank) depends critically on the quality of the reward signal: when the reward is dense and meaningful, maintaining adaptability is beneficial; when it contains structured uninformative segments arising from catastrophic failure modes, lower adaptability acts as a form of noise resistance.

#### Critic performance

Finally we evaluate the value network performance in terms of prediction error (MSE) between the predictions $f_\phi$ and the targets $\hat{Q}$. We also measure and study the fluctuation of the bias and the variance. In order to measure how $\alpha_{\text{ev}}$ values influence the critic performance, we cannot just compare the values of our metrics across different $\alpha_{\text{ev}}$ values. Indeed, as $\alpha_{\text{ev}}$ modifies the value network, it modifies the policy, which modifies the distribution used to train the critic. As a consequence, a value network that leads to a better performing policy may have a worse error than a value network that leads to poor performance as they are evaluated on different trajectories. In order to compare the EVarEst critic with the baseline MSE critic, we instead evaluate and train them on the same dataset. To do so, we use EVarEst for the usual value network, and we add an additional value network in parallel, which has no impact on the policy, trained using the MSE on the same data as the first critic. We can then compare the metrics for both models while they have been exposed to the same data and are evaluated on the same data. We are comparing relative difference to the baseline critic. For more details, please refer to Appendix A.3. It is important to note that using this protocol, two critics with the same objective may not have the same performance as (1) they are initialized independently, and (2) the EVarEst critic is used to update the actor and the MSE critic is not.

To study how using EVarEst instead of the MSE changes the value network performance in terms of prediction error, we compute the relative difference between a value network trained using EVarEst and another trained using the MSE objective, for each of the three previously mentioned metrics: MSE, Bias$^2$, and VRE. Letting $m_t^{EVarEst} = \text{metric}(f_\phi^{EVarEst(\alpha_{\text{ev}})}(s_t, a_t), \hat{Q}(s_t, a_t))$ and $m_t^{\text{MSE}} = \text{metric}(f_\phi^{\text{MSE}}(s_t, a_t), \hat{Q}(s_t, a_t))$, the relative difference is:

$$\Delta_{\text{rel}}(\text{metric}, \alpha_{\text{ev}}) = \frac{100}{N} \sum_{t=1}^{N} \frac{m_t^{EVarEst} - m_t^{\text{MSE}}}{|m_t^{EVarEst}| + |m_t^{\text{MSE}}|}.$$

A positive value of the relative difference means that EVarEst leads to a value network with higher metric (MSE, Bias$^2$ or VRE) than a value network trained using the MSE, a negative value is the opposite. Note that the denominator uses the sum of absolute values rather than a single reference metric; this choice guards against near-zero denominators when both predictions are close to the target, at the cost of compressing the reported percentages by up to a factor of two compared to a standard relative difference. The sign and direction of all effects are unaffected. We concentrate on SAC in this section, while results for PPO can be found in Appendix A.2.3.




**Figure 5.6** – Relative difference between EVarEst-SAC and baseline MSE value networks for value estimation metrics (MSE, Bias$^2$ and VRE) averaged on the last $10^5$ timesteps with different $\alpha_{\text{ev}}$ values

In Figure 5.6, the MuJoCo results reveal a strong and clear trade-off: lower $\alpha_{\text{ev}}$ substantially reduces VRE (relative difference approaching $-100\%$) at the cost of increased Bias$^2$ (approaching $+100\%$). Importantly, the overall MSE (blue line) remains close to zero across all $\alpha_{\text{ev}}$ values, indicating that the variance reduction largely compensates for the bias increase in terms of aggregate prediction error. This confirms that EVarEst successfully re-distributes prediction error from variance toward bias, as intended by the re-weighting. On AdroitHand, the picture is less consistent: for AdroitHandPen in particular, lower $\alpha_{\text{ev}}$ produces a *positive* VRE relative difference, meaning EVarEst exhibits higher residual variance than the MSE baseline. This is consistent with the noise-fitting interpretation from Section 5.5.4: a high-rank network that

tracks uninformative returns exhibits larger error variance than a lower-rank, more biased network.

### Robustness

Finally, in order to study the robustness of our method with respect to the policy non-stationarity, we want to measure how much a change in policy will impact value estimations. To do so, we consider the difference between successive loss values. If, for the same variation in policy, one objective leads to a smaller difference in loss, then we say it is more robust to policy variation than another. We follow the same methodology as in Section 5.5.4 with parallel value networks to compare different value objectives. We compare the difference in successive loss between a value network with EVarEst objective and another with the MSE objective. This allows both estimators to share the same policy non-stationarity, removing the need to compute the distance between successive data distributions generated by the policy. We analyze Mujoco tasks in this section, AdroitHand tasks can be found in Appendix A.2.5.

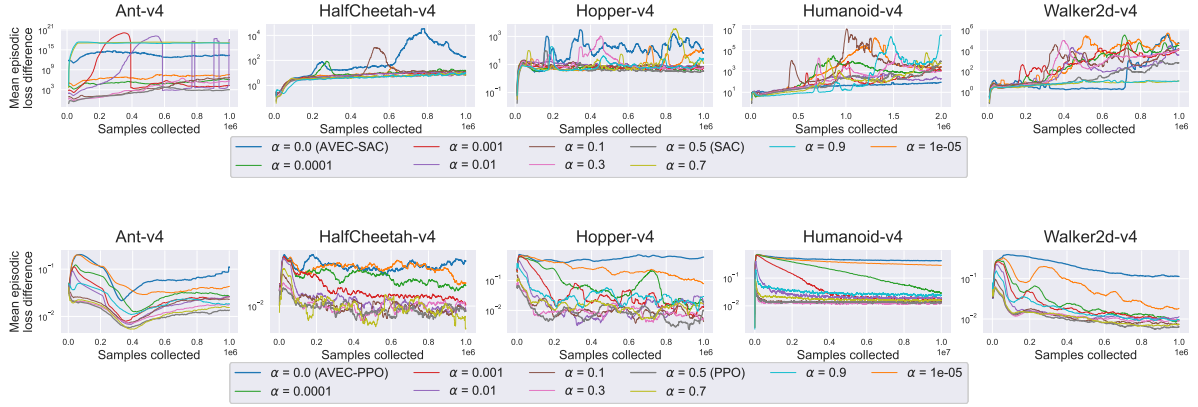



**Figure 5.7** – Relative difference between EVarEst and baseline MSE for value loss over training for different $\alpha_{\text{ev}}$ values on Mujoco tasks (Top: SAC, Bottom: PPO)

In Figure 5.7 we can observe that lower values of $\alpha_{\text{ev}}$ tend to lead to higher variations in loss across successive policy updates, i.e., lower stability by our proxy measure. This is the opposite of the robustness effect we originally hypothesized.

However, this negative result calls for reinterpretation rather than simple dismissal. Loss variation across policy updates is not inherently harmful: it is a natural consequence of a value network that remains *responsive* to distribution shifts. A network that maintains a rich, high-rank representation (as promoted by lower $\alpha_{\text{ev}}$) will show larger loss changes when the policy updates, precisely because it is actively tracking the new distribution. By contrast, a network that has undergone capacity loss may appear stable simply because it has stopped meaningfully adapting to policy changes — its loss does not vary much because its predictions no longer change much. In this light, the loss stability we set out to measure is a misleading

proxy for the property we actually care about: the network's capacity to fit successive policies well over time. The feature rank results from Section 5.5.4 provide the more direct measure of that capacity, and they paint the opposite picture from the loss stability results. Together, these two measurements suggest that criterion (1) of Section 5.4.1 (maintained plasticity) is more relevant to performance than criterion (2) (loss stability), and that what we labeled as lower robustness is better understood as higher sensitivity: the other face of adaptability.

## 5.6 Conclusion

In this chapter, we introduced, evaluated, and analyzed a new value estimation objective, EVarEst, which is designed to better address the inherent non-stationarity in reinforcement learning compared to the traditional MSE objective. By incorporating a hyperparameter, $\alpha_{\text{ev}}$, to balance the bias-variance trade-off of the MSE, EVarEst effectively penalizes the classical MSE, allowing for adaptive control over the VRE. Importantly, this requires minimal modifications to existing algorithms.

We demonstrated that EVarEst, with $\alpha_{\text{ev}}$ adapted to each task, significantly improves performance across a range of tasks, achieving an average improvement over the MSE of 13.29% on AdroitHand and Mujoco tasks for SAC and 28.7% for PPO. This suggests that the classical MSE value estimation objective is often suboptimal, and that task-specific adaptation of $\alpha_{\text{ev}}$ leads to better performance in reinforcement learning.

The experimental analysis of EVarEst's impact on the learning process yields a nuanced and in part unexpected picture. Our original hypothesis was that lower $\alpha_{\text{ev}}$ (more VRE focus) would improve robustness to non-stationarity, measured as stability of the loss across successive policy updates. This hypothesis was not supported: lower $\alpha_{\text{ev}}$ values consistently produce *higher* loss variation. However, the feature rank analysis points toward a different and more informative mechanism: lower $\alpha_{\text{ev}}$ maintains higher feature rank throughout training, preserving network plasticity and the capacity to adapt to new policy distributions. We propose that the elevated loss variation is not a sign of harmful instability, but the signature of a responsive value network that actively tracks policy changes — as opposed to a collapsed, low-rank network whose apparent stability reflects degraded adaptation rather than genuine robustness.

The performance results further reveal that the optimal $\alpha_{\text{ev}}$ depends strongly on the benchmark. On MuJoCo locomotion tasks, lower $\alpha_{\text{ev}}$ is almost universally beneficial for SAC: the reward signal is dense and well-structured, and the main challenge is indeed policy non-stationarity. In this regime, maintaining high feature rank through VoRE focus pays off consistently. On AdroitHand, the picture is more complex: on most tasks the pattern holds, but AdroitHandPen specifically favors high $\alpha_{\text{ev}}$ (near full MSE). The feature rank analysis explains this reversal: AVEC ($\alpha_{\text{ev}} = 0$) achieves the highest feature rank on AdroitHandPen but the worst performance.

The culprit is reward signal quality — when the agent drops the pen, the episode continues with a long stream of uninformative negative returns. A high-rank, highly adaptive network faithfully fits this noise, building a value function that no longer serves the policy. A more biased, lower-rank network resists this noise and provides a more useful training signal. Together, these findings suggest that $\alpha_{\text{ev}}$ controls a trade-off between adaptability and noise resistance, and that the optimal operating point is determined not by non-stationarity alone, but by the interaction between policy shift rate and reward signal quality.

We argue that the standard MSE objective, with its implicit equal weighting of bias and variance, is not the natural choice for the non-stationary RL setting, and that a more deliberate trade-off consistently improves performance. EVarEst provides the parametric flexibility to express this trade-off with minimal modifications to existing algorithms. The principal open problem this work surfaces is how to select $\alpha_{\text{ev}}$ without exhaustive search. The feature rank trajectory is a candidate online signal: sustained high rank suggests that VRE focus is beneficial, while high rank combined with degrading performance may indicate noise fitting rather than useful adaptation. We believe that further investigation of the relationship between task properties (reward signal structure, policy shift rate, failure mode topology) and the optimal $\alpha_{\text{ev}}$ is a productive direction for improving value estimation in deep RL.

Addressing robustness is one axis of this thesis; the other is efficiency. Even the best algorithm is of limited use if training takes too long to be practical, whether in research or in an industrial context. Chapter 6 examines how parallelization, both at the algorithmic and the implementation level, can dramatically reduce wall-clock training time and, as a side effect, improve policy robustness through exposure to diverse experience.

# Part III

# Efficient Reinforcement Learning

# Chapter 6

# Parallelization in Reinforcement Learning

> « τὸ δὲ πᾶν οὐκ ἔστιν ὥσπερ σωρὸς ἀλλ᾽ ἔστι τι τὸ ὅλον παρὰ τὰ μόρια. »
> *"The whole is something over and above its parts, and not just the sum of them all."*
>
> Metaphysics — **Aristotle** (c. 384–322 BC)

The Rainbow algorithm (Hessel, Modayil, et al., 2018) combines six independent extensions to DQN into a single agent. Shengyi Huang et al. (2022) identify 37 implementation details, most unreported in the original paper, that together account for a large share of PPO's performance advantage over its predecessor. These are not isolated examples: they illustrate a broader trend in which state-of-the-art deep RL algorithms accumulate components and decisions, each carrying its own computational cost and tuning surface. As algorithms grow in depth, so does their execution time, and with it the budget required to run a statistically meaningful experiment.

Computational cost also sets an entry barrier. A method that requires data-center resources to reproduce is, in practice, a method that only a handful of institutions can study. Keeping experiments tractable on a single high-end consumer GPU, or at most a small cluster, is a precondition for an accessible research community. Parallelization is one of the most direct tools available for that purpose, and is the subject of this chapter.

## Contents


**6.1 Algorithmic parallelization** . . . . . 86
**6.2 Implementing parallelization** . . . . . 92
**6.3 Conclusion** . . . . . 103

## 6.1 Algorithmic parallelization

### 6.1.1 Overview of parallelization scheme in deep RL

In supervised deep learning, parallelism is synonymous with efficiency. Forward and backward propagation in artificial neural networks is highly parallelizable due to the large amount of highly dimensional vector operations. It goes without saying that being capable of running operations in parallel, rather than in sequence, can be highly beneficial for time efficiency. In deep reinforcement learning, as the same neural networks are used, this parallelization extends as well. Naturally, as for any relatively time consuming process, researchers have looked into making deep reinforcement learning algorithms faster beyond just parallelization inside the neural networks. To study the potential vectors of parallelization inside deep RL algorithms, we will begin by describing the common operations in a deep RL algorithm. Although the actual modules will vary depending on the algorithms, conclusions will still apply to most of them.

**Algorithm 3** Generic Deep RL Backbone

**Initialize:** Main $\theta$, target $\theta^- \leftarrow \theta$, and Buffer $\mathcal{D}$
**Initialize:** Learning rate $\eta$, Budget $B$, Rollout size $T$, Epochs $N$
Receive initial state $s_0$
**for** $t = 1$ **to** $B$ **do**
  **for** $k = 1$ **to** $T$ **do** // Collection
    Select $a_k \sim \pi_\theta(s_{curr})$
    Execute $a_k$, observe $r_k, s_{next}$
    Store $(s_{curr}, a_k, r_k, s_{next})$ in $\mathcal{D}$
    $s_{curr} \leftarrow s_{next}$
    **if** $s_{curr}$ is terminal **then**
      Restart environment: $s_{curr} \leftarrow ss_0$
    **end if**
  **end for**
  **for** $n = 1$ **to** $N$ **do** // Update
    Sample minibatch $\mathcal{B} \sim \mathcal{D}$
    $\theta \leftarrow \theta - \eta \nabla_\theta J(\theta, \mathcal{B})$
    $\theta^- \leftarrow \tau\theta + (1-\tau)\theta^-$
  **end for**
**end for**

As previously discussed, a vast majority of these operations is already parallelized as they stem from deep learning standard procedures (Policy network inference and updates, sampling data). The specificity of deep RL comes from the need to collect data as we go. This collection step usually represents a large fraction of the overall training duration. A clear candidate for making deep RL algorithms quicker is thus to make the data collection faster. To do so, one might be tempted to use parallelization, just like the other modules. Parallelization however, is not as natural for data collection from an environment as it is for deep learning operations. A

first issue arises from trying to parallelize RL: RL is inherently sequential, as action selection depends on the state, which itself depends on the previous state, which depends on the previous action, etc. Therefore, it is only possible to collect one transition at a time from one environment. An obvious solution is to collect data from multiple environments at the same time, allowing for parallelization of these interactions. This creates two new issues to solve before parallelization can be achieved:

- Obtaining multiple instances of an environment is not trivial (e.g., multiple identical robots). In practice this is mostly restrictive to simulations, which can be easily copied.
- Collecting a single trajectory of length $N$ from a single environment does not contain the same information as collecting $K$ trajectories of size $\frac{N}{K}$ from $K$ environments (at least in the episodic setting). An extreme example of that would be for episodes of size $K$, in this setting the $K$ parallel environments would provide 1 transition, while a single environment would have yielded a full episode. Learning from a full episode would likely provide more learning signal than learning $K$ 10 first transitions.

As a consequence, parallelization can be of interest in tasks where cheap simulations are available, and with the added concern of adapting the data collecting process to these parallelized environment setting.

### 6.1.2 A brief history of parallelization applied to deep RL

**Early distributed deep RL** PPO (Schulman, Wolski, et al., 2017) is probably the most commonly used baseline in deep RL, and it does use multiple parallel environments. However, parallelization is not its primary contribution: the original paper focuses on the clipped surrogate objective and multi-epoch minibatch updates, and the use of multiple environments (typically 4 to 8) is treated as a standard implementation detail. The earliest algorithm explicitly designed around large-scale parallelism is Gorilla (A. Nair et al., 2015), which distributes DQN learning across hundreds of workers. Its limitation is also its defining feature: it required data-center computing power, making it inaccessible outside a handful of institutions. For research reproducible on a single machine, the more relevant starting point is A3C (Volodymyr Mnih, Badia, et al., 2016).

A3C runs multiple independent actor-learners in parallel on different CPU cores. The goal is not only speed, but data diversity. A key difficulty in deep RL is that training data is inherently temporally correlated and non-stationary: consecutive transitions come from the same trajectory and destabilize stochastic gradient updates, while the data distribution shifts as the policy improves. The link to the i.i.d. assumption runs through the optimizer: stochastic gradient descent is known to converge under the assumption that mini-batches are approximately independent draws from a fixed distribution (Bottou, Curtis, and Nocedal, 2018). Correlated, non-stationary RL transitions violate both conditions. Sutton and A. Barto (2020) further

identify the combination of function approximation, bootstrapping, and correlated data as the *deadly triad*, a regime in which value estimates can diverge even when each component is individually sound. The goal of any decorrelation mechanism is therefore to bring training conditions as close as possible to the approximately independent-sample regime that supports convergence (Volodymyr Mnih, Kavukcuoglu, D. Silver, Graves, et al., 2013; Volodymyr Mnih, Badia, et al., 2016).

Before A3C, the standard answer to sample correlation was the replay buffer: transitions are stored in a large pool and sampled randomly, breaking their temporal ordering (Volodymyr Mnih, Kavukcuoglu, D. Silver, Graves, et al., 2013). A3C proposes an alternative that is also an opportunity to emphasize a recurring point: parallelization is not just an execution speedup. Running multiple independent environments simultaneously reduces correlation not by discarding data, but by collecting data that was never correlated to begin with. Its synchronous variant A2C is usually preferred in practice, since synchronous algorithms are better suited to GPU execution and using a GPU is more beneficial in terms of throughput than managing multiple asynchronous workers (Yuhuai Wu, Mansimov, Liao, et al., 2017).

**Emerging properties** While A3C was limited to 'only' up to 32 environments in parallel, other asynchronous methods like Importance Weighted Actor-Learner Architecture (IMPALA, Espeholt et al. (2018)) or Evolution Strategies (Salimans et al., 2017), exploiting multiple workers did manage to scale to hundreds of parallel learners. However, for the reasons stated above, and for the fact that it was easier to use a single GPU that multiple distributed workers, the community shifted to synchronous and mostly single-device algorithms. Before actually achieving significant time gains through parallelization, algorithms like GA3C (GPU-based A3C) attempted to do so but were limited by the CPU-GPU bottleneck. We will discuss this issue in the following section regarding implementations and hardware. The first work to drastically speed up learning thanks to parallelization was "Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning" by Rudin et al. in 2021, where they managed to increase the efficiency from previous work by orders of magnitude: going from multiple days of training to twenty minutes, on a single GPU. Once again, while parallelization did allow for a faster collection of samples, it is not the only impact it had on learning. A key phenomenon reported in this work is that by having a thousands of environments to collect data from, the diversity in the data yields policies that are more robust[1] than if they had been trained on a similar number of transitions from a few environments. Beyond that, the authors could also introduce additional artificial variety in these environments in the form of slight variations in the tasks (usually referred to as Domain Randomization Tobin et al. (2017) e.g., different friction coefficients, different terrain) and combined with curriculum learning to not introduce this increased task variety (and thus difficulty variety) all at once and hurt learning. Not only

[1] in the sense that they generalize better to unseen situations

did this process help accelerate the training, but it also improved the robustness of the learned policy when transferring from simulation to the real world (referred to as Sim-to-real X. B. Peng et al. (2018) transfer), which is usually a staple difficulty of cases where training is done in simulations. This work is a perfect example of the emerging properties that are offered by massive parallelization.

**Off-policy parallelization** The robustness gains from massive parallelization are not limited to on-policy methods. MPO was used with 5000 parallel actors to control a nuclear fusion tokamak reactor (Degrave et al., 2022), a striking illustration of how diversity of experience can stabilize learning in safety-critical applications. The question then arises: can off-policy algorithms benefit equally from massive parallelization?

Switching to a highly parallel setting is straightforward for on-policy algorithms, since each collected rollout is immediately used and discarded. Off-policy algorithms are more complex: they rely on a replay buffer whose purpose is precisely to provide a diverse, decorrelated dataset. At thousands of parallel environments, the buffer is refilled so rapidly that it ends up containing only the most recent transitions, losing the temporal diversity that justified its use in the first place. Meanwhile, off-policy algorithms require finished collection before starting updates, limiting opportunities for concurrent execution.

Parallel Q-Learning (PQL, Zechu Li et al. (2023)) (Zechu Li et al., 2023) showed that this tension can be resolved: by exploiting the off-policy property to interleave updates and collection, it achieves better sample efficiency and wall-clock time than PPO at matched environment counts. However, in the same paper, parallelized SAC underperforms parallelized PPO in all metrics, suggesting that not all off-policy algorithms benefit equally from the massively parallel regime. PQL also requires asynchronicity, making it impractical on a single GPU.

Subsequent work has pushed further. Parallelized Q-Network (PQN, Gallici et al. (2025)) (Gallici et al., 2025) showed that Layer Normalization (J. L. Ba, Kiros, and Hinton, 2016) can replace both the replay buffer and the target network, yielding a stable off-policy algorithm with no asynchronicity requirement. FastTD3 (Seo, Sferrazza, Geng, et al., 2025) demonstrated that simple design choices (batched updates, a distributional critic, careful hyperparameter selection) are sufficient for efficient parallel TD3. And while the PQL paper found SAC underperforming, later work (Seo, Sferrazza, J. Chen, et al., 2025; Raffin, 2025; Shukla, 2025) showed the opposite with proper tuning. The current state of the field[2] suggests that off-policy methods, rather than on-policy ones, are leading in the massively parallel setting.

[2] As of early 2026.

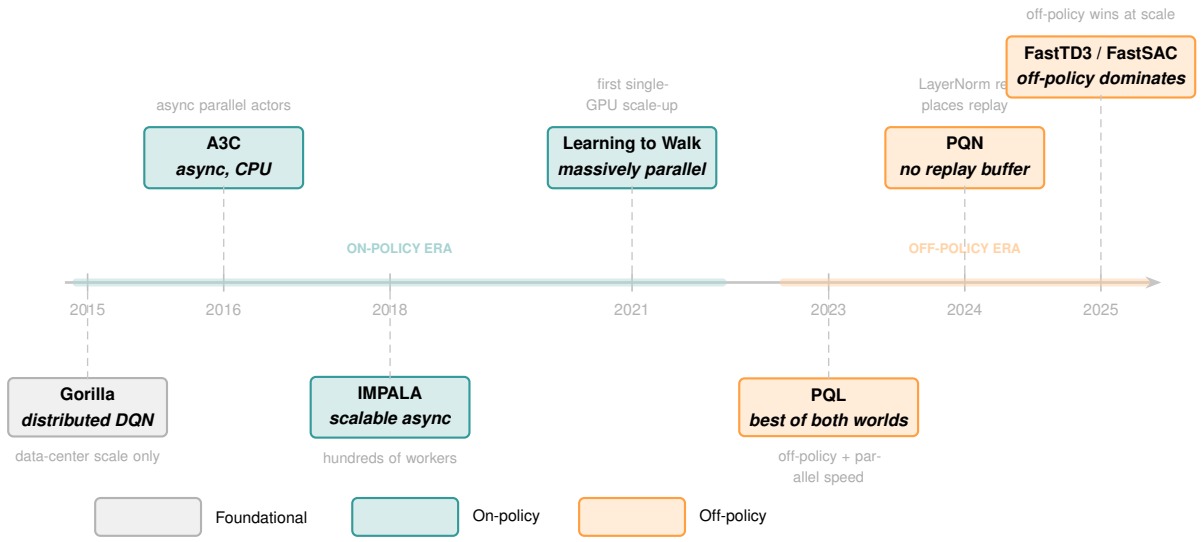



**Figure 6.1** – Key milestones in the history of parallelization applied to deep RL. Color indicates whether the algorithm is on-policy (teal) or off-policy (orange). The progression reflects a shift from distributed CPU infrastructures toward massively parallel single-GPU pipelines, and more recently from on-policy dominance toward off-policy methods.

**Model-based and MPC approaches** All the aforementioned methods rely on an explicitly parameterized policy. A different route is to frame the policy as the solution of an optimization problem, as in MPC. This family of methods is particularly relevant to industrial control, where PID and MPC controllers are the dominant approach and practitioners often have physical models available. TD-MPC2 (Hansen, Su, and X. Wang, 2024) bridges this world and deep RL by adopting Model Predictive Path Integral (MPPI, G. Williams et al. (2016)), a sampling-based MPC method, and leverages parallelism by massively sampling possible trajectories at planning time. This work sits at some distance from the rest of the chapter, which focuses on actor-critic algorithms; it is included here because the connection between model-based control and massively parallel RL is directly relevant to industrial applications, even if it is not a direction pursued in this thesis.

**Industrial perspective**

The industrial setting studied in this thesis may appear at first to require sample efficiency above all else: interacting with the furnace is expensive, and even simulations are slow. However, the approach demonstrated by Rudin et al. (2022) suggests an alternative: a fast, coarse simulation run with massive parallelization can produce policies robust enough to transfer to the real system. We believe this route is applicable to our industrial context, and discuss it further in the conclusion of this chapter.

### Hybrid Sim2Real RL Architecture

In other words, leveraging massive parallelism can help mitigate the misspecification of simulation, and shift a sample efficiency focused setting to a wall-clock time setting. We argue that this is already what has happened in robotics, where the use of sample efficient algorithms like SAC is the gold standard, but its hegemony was challenged by the ease of parallelization of PPO and its advantage in faster time to learn policies that work on real robots. It is important to note that in the setting where the agent is trained in massively parallel simulation and then applied to the real-life system, although the results in "Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning" showed that the policy learned the desired behavior, it was still not as good as if the policy was subsequently fine-tuned directly on the real system. Hence, in this later phase where real-life interactions are required, sample efficiency is still the gold standard. This leads to considering a pipeline with a massively parallel wall-time focused algorithm learning on a fast simulation, which is then fine-tuned with a sample-efficient algorithm on the real system.

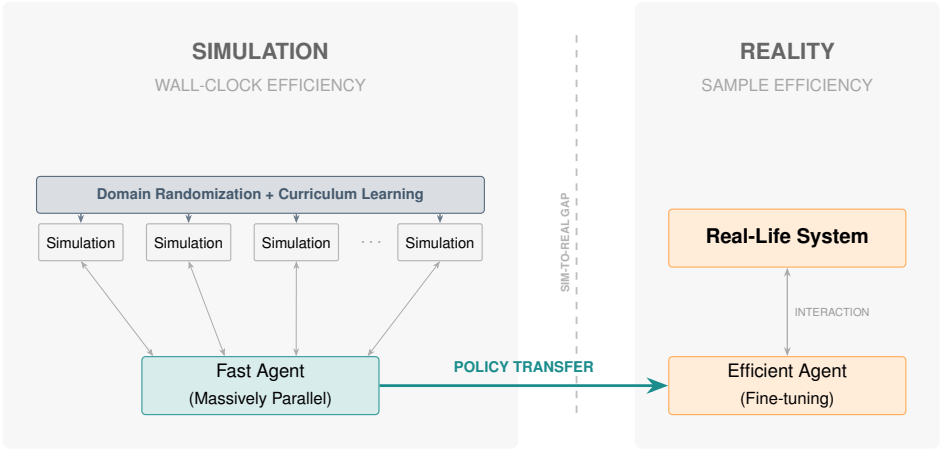



**Figure 6.2** – Hybrid Sim2Real RL Architecture: The left side of this diagram depicts the massively parallel training of a fast agent. This allows for rapid training as well as building robustness w.r.t. model misspecification. The policy learned in simulation is then transferred to a more sample-efficient agent for fine-tuning on the real system.

However, that opens the question of finding either (1) an algorithm that works well in both settings and can be used end-to-end, or (2) to find a way to hand over the learned behavior from the simulation agent to the real-life agent without losing the robustness to environment misspecification. An approach for route (1) may be to use SAC as it is both sample efficient in the single environment setting, and can be made time efficient in the massively parallel setting; however, it remains untested to our current knowledge. We believe there is a lot of potential for complex real-life tasks in this approach, as they are often limited by the accuracy or availability

of the simulation, which could be relieved by the parallel setting. More work is required to experiment with this pipeline and the best algorithm candidates.

**Algorithm 4.** Hybrid Wall-Clock to Sample-Efficient (W2S) Pipeline

**Input:** Simulations $\{S_1, \ldots, S_N\}$, Real system $R$, Budgets $T_{\text{sim}}, T_{\text{real}}$
**Initialize:** Fast Agent $\pi_\phi$, Efficient Agent $\pi_\theta$

**// Phase 1: Massively Parallel Pre-training (Wall-Clock Focus)**
**for** iteration $t = 1, \ldots, T_{\text{sim}}$ **do**
  **for all** Environments $S_i$ **parallel do**
    Step environment $S_i$ and collect experience $\mathcal{E}_i$
    **if** Environment $S_i$ terminates **then**
      Update $\mathcal{D}_t$ and $\mathcal{C}_t$ for $S_i$ ▷ Domain Rand. & Curriculum
      Reset $S_i$ with new task parameters
    **end if**
  **end for**
  Optimize $\pi_\phi$ using aggregated experience $\{\mathcal{E}_1, \ldots, \mathcal{E}_N\}$
**end for**

**// Phase 2: Policy Hand-over**
Transfer learned features and behaviors from $\pi_\phi \to \pi_\theta$

**// Phase 3: Real-World Fine-tuning (Sample Efficiency Focus)**
**for** step $k = 1, \ldots, T_{\text{real}}$ **do**
  Execute action $a_k \sim \pi_\theta(s_k)$ on real system $R$
  Observe transition $(s_k, a_k, r_k, s_{k+1})$
  Store transition in sample-efficient replay buffer $\mathcal{D}$
  Optimize $\pi_\theta$ using gradients from $\mathcal{D}$ ▷ e.g., SAC update
**end for**
**return** Final fine-tuned policy $\pi_\theta$

## 6.2 Implementing parallelization

After what we discussed of the history of parallelization in deep RL, one might comment that if parallelization seems to be so beneficial, why it did not come earlier in the history of deep RL, or why it is not more broadly used. The first, and most important answer, is that as mentioned previously, parallelization is not always possible. When interacting with real-life systems without a possibility for simulation, there is no realistic way to parallelize other than having multiple instances of the real-life system. This approach may be either impossible or too

expensive. The second and more practical explanation for the complexity of implementing and using parallelization in deep RL are hardware considerations. Strictly speaking, the parallel aspect of the algorithms we discussed previously is only a requirement for speed, as it is possible to collect from different environment in sequence rather than in parallel. However, in doing so, we lose the main appeal of parallelization, or worse, the capability to run those experiments in the first place due to unrealistic computation time. In order to effectively take advantage of those independent environments, we need to actually make them run in parallel. More than making them run in parallel, we also want the algorithm to efficiently run in parallel, to yield a speed increase when compared to a sequential version. In practice that usually boils down to having the implementation of the algorithm and the environment running on an accelerator (we will only talk about GPUs in this work, but the same holds for TPUs).

Classic execution of programs on a computer happens on the CPU. Making code run on a GPU requires two main extra-steps, (1) code needs to be adapted to the GPU (as the basic instructions are not shared with the CPU), and (2) the instructions and data need to be sent to the GPU and back. Making environments parallelizable (1) means implementing them in a way that can be executed on GPU. This is the first major hurdle in parallel deep RL. Even if the environment is not actually capable of running in parallel, and will thus be executed on a single core of the CPU; one might want to at least have the rest of the algorithm, which we know for the most part to be capable of parallelization, to run in parallel on a GPU. However, this is where the second major hurdle of parallel deep RL comes in, (2) the CPU-GPU bottleneck.

### 6.2.1 CPU-GPU bottleneck

In a classic setting where the environment cannot be run on GPU, and is therefore executed on CPU (maybe parallelized across cores in best case scenario), and one would like to have the deep learning parts of the algorithm on GPU (as they are usually implemented using a deep learning framework like Tensorflow or Pytorch which provides the possibility to execute on GPU); one might realize that doing so will more often than not impair the execution speed. In most online-RL algorithms, the agent collects some number of transitions from the environment (which we will assume is unique here), and will then use the collected data (from the collected rollout or a buffer) to update its policy. As the neural network(s) used to represent the policy are executed on the GPU, the instructions and data need to be transferred to the accelerator. The instructions are loaded once at the beginning of training; however, the data still has to be loaded onto and then unloaded from the accelerator at each data collection step. This data moving operation is relatively costly when compared to the actual training on GPU, and since it needs to be done for every update and every action selection, it piles up very quickly.

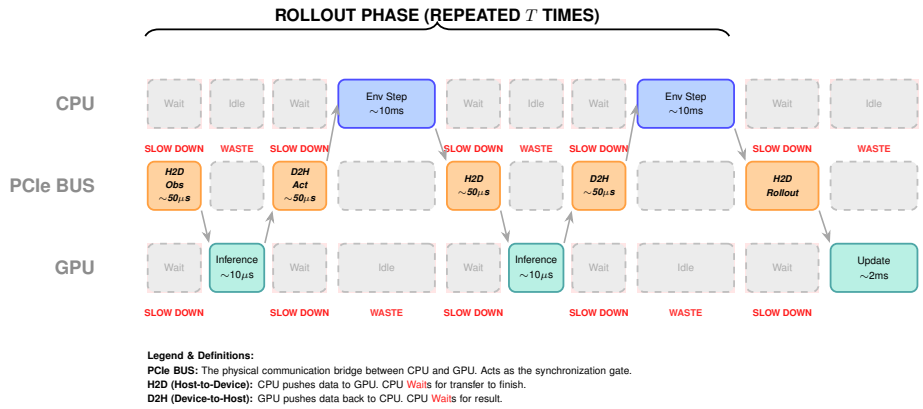



**Figure 6.3** – Schematic representation of temporal bottlenecks in the CPU-GPU pipeline. Timings are provided for illustrative purposes.

This is the CPU-GPU bottleneck: data transfer between the two devices becomes the leading cause of execution time. Figure 6.4 shows a concrete measurement of this cost on a standard RL workload, comparing a pure CPU pipeline against a hybrid CPU+GPU one where the neural network runs on GPU but the environment stays on Central Processing Unit (CPU).

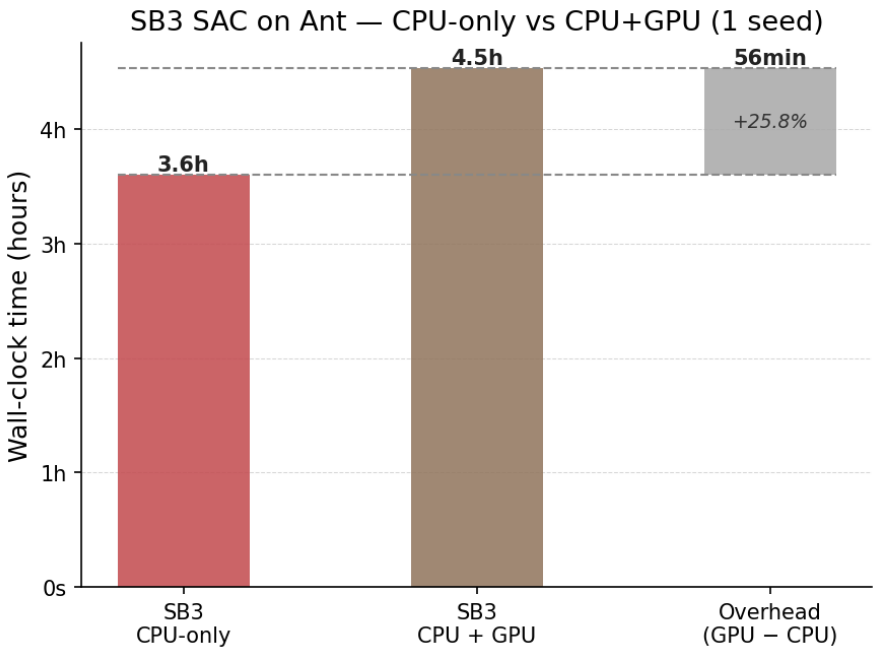



**Figure 6.4** – Wall-clock time comparison between a pure CPU pipeline and a hybrid CPU+GPU pipeline on the same RL workload. The bottleneck induced by host-device transfers negates most of the expected GPU benefit. Measured on SB3 SAC on Gymnasium (MuJoCo) Ant with an Nvidia L40S GPU and AMD EPYC 9454 48-Core Processor CPU

To prevent this loss entirely, all operations must run on the same device. Staying on CPU is one option, and can be optimized further through parallel vectorization (e.g., envpool (Weng et al., 2022)). The more powerful alternative is to run the entire training loop on the GPU: eliminating the bottleneck rather than tolerating it. The gains from this approach can be dramatic: speedups of 100× to 1000× over a pure CPU pipeline have been reported when environments are ported to full GPU execution (Makoviychuk et al., 2021; Freeman et al., 2021). The question is therefore how to actually port the environment to GPU.

### 6.2.2 The complexity of GPU code

Asking how to port an environment to GPU (or TPU) first requires to understand how code is executed on CPU. Assuming the environment is written in a programming language, either high-level or low-level language, the code is still here to represent instructions in a way that can be understood by humans. To be executed on a device, which can only execute binary code, the code usually goes through two steps. First it is interpreted or compiled from programming language (e.g., Python (Van Rossum and Drake, 2009) or C++ (ISO, 2020)) to assembly. This assembly code is then translated to binary code. Assembly code is device-architecture specific (e.g., x86, ARM), so every different architecture needs to be handled by the interpreter/compiler. GPUs are a different kind of device which requires a specific set of instructions. GPUs are different from CPUs as their architecture is specifically made to handle graphical operations more efficiently. In practice, this essentially boils down to making Single Instruction, Multiple Data (SIMD) instructions more efficient, which means making the same computation on a large number of data at the same time. These types of operations are the bread-and-butter of forward and backward propagation in neural networks due to the important number of large matrix computations. The instructions for this kind of architecture are therefore different from the ones for CPU architectures. Even worse, programming languages are different, e.g., Compute Unified Device Architecture (CUDA, NVIDIA Corporation (2024)) C++ for Nvidia GPUs and HIP (Advanced Micro Devices, 2024) for AMD GPUs.

Therefore, to write environments to work on GPUs, one must be able to generate the assembly code for GPUs which gives results equivalent to the CPU one (as we do not want them to be identical, due to their difference in architecture). This usually requires a different programming skillset (e.g., learning C++ and CUDA C++) than the one deep RL researchers usually have. As this skillset can be considered much harder to learn than the typical Python skillset, this is a major hurdle to implementing RL environments for GPUs. Nonetheless, examples of such implementations exist, the most important one being Isaac Gym (Makoviychuk et al., 2021) which uses CUDA C++; however, it is limited to robotics environments.

### 6.2.3 Jax as a complexity middleground

**Jax strengths** A middleground solution between Python and Cuda C++ is Jax (Bradbury et al., 2018), a Python numerical computation library/framework. Jax combines most of the ease of coding from Python, with the efficiency of compiled lower-level languages like C++, with the extremely important added benefit of sharing the same code for all platforms (CPU, GPU or even TPU). This means that without having to write (and maintain) different versions of the same environment, once it is written in Jax, it will seamlessly work on all platforms[3]. Most of JAX's syntax is inherited from NumPy (Harris et al., 2020) through the use of 'jax.numpy', this is an advantage as this is very commonly used in environment implementations and machine learning in general. Therefore, if one already has a Python implementation of their environment that heavily relies on numpy, half of the translation work to Jax only requires replacing 'numpy' by 'jax.numpy'.

To achieve this cross-platform execution, Jax compiles the code (as opposed to Python's interpreted nature) by executing the code and replacing the actual data with so called tracers, special objects which collect the operations during execution. The collected operations are then merged into a computational graph representing the program. This graph is then processed to an intermediate representation called Jaxpr (an abstraction of the collected operations). This Jaxpr is then lowered to StableHLO. This optimized representation is then translated by the Accelerated Linear Algebra (XLA, Leary and T. Wang (2017)) compiler to High-Level Operations (HLO) IR. It is then translated into Low-Level Virtual Machine (LLVM, Lattner and Adve (2004)) IR (Lattner and Adve, 2004). LLVM IR can be then compiled for any of the platforms supported by LLVM: different architectures of CPUs or GPUs (e.g., Parallel Thread Execution (PTX) for Nvidia GPUs).

[3]That are compatible with XLA, which is not very restrictive

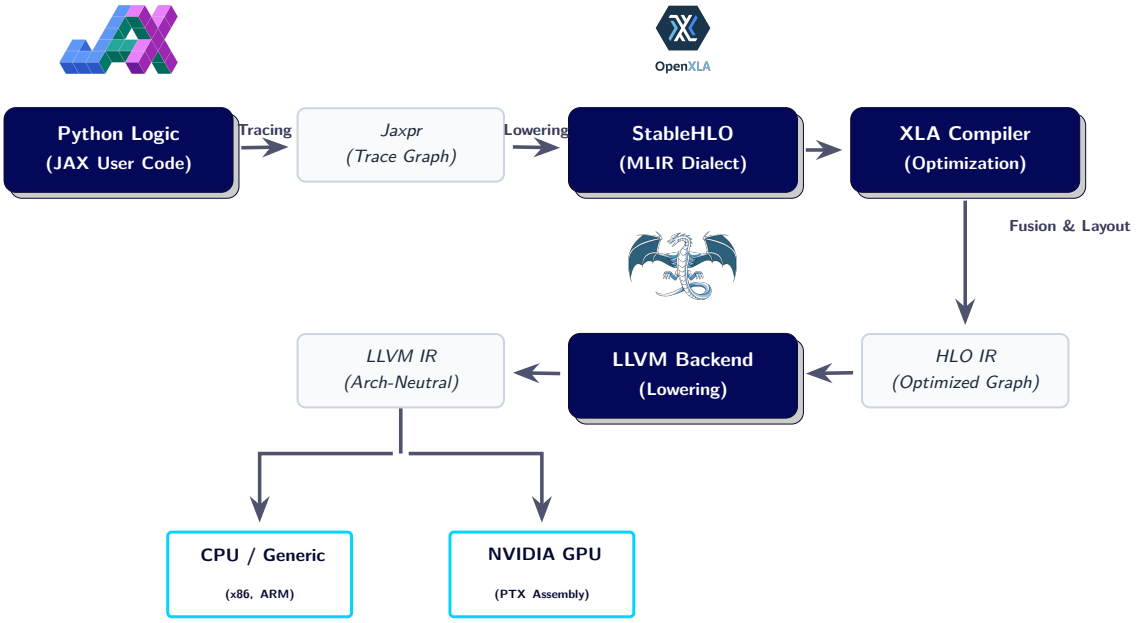



**Figure 6.5** – Schematic representation of the Jax compilation process

The compilation style used by JAX is called Just-In-Time (JIT) compilation, as it first interprets the Python code (as Python normally would) with tracers, and then uses the graph collected through the tracers to compile the actual program. Therefore, it is not compiling the source code itself, but compiling during interpretation; hence, it is not generating a standalone compiled program. Being compiled as it is executed leads to the "just-in-time" aspect of JIT compilation. Technically, JAX does not need to compile code to execute it. However, by avoiding compilation, most of the advantages of JAX are lost. It is therefore crucial for performance to make most, if not all, of the code "jittable".

**JAX Weaknesses** While JAX provides significant performance gains, it comes at an increased programming complexity cost. This cost can be broken down into several key challenges:

**Compilation** The first paradigm shift is moving from interpreted to compiled. It is worth noting that compilation is not a concept specific to JAX: languages such as C or C++ (ISO, 2020) have long benefited from compiled execution, and the performance advantages it provides are well established. For practitioners coming from Python, however, this may feel unfamiliar, as Python is an interpreted language and these compiled-language idioms are less common in the machine learning ecosystem. What makes JAX notable is that it brings compiled-language performance to a Python-based workflow without requiring a full language change. This transition is a double-edged sword: while compilation allows for better efficiency down the line through the fusion of operations into efficient blocks, it also makes writing code harder than with standard Python. This added coding complexity comes notably from a harder time

accessing the values of arrays during debugging because JAX works with tracers, special objects that collect operations along the program by mimicking real data. No actual data is processed during compilation. Therefore, if the program does not compile, it is not possible to observe the values that may lead to those problems. It is, however, possible to do so during execution using JAX's own debug tools (which requires the program to compile first).

**Static Typing and Shapes** The second most important shift is moving from dynamic to static typing; this includes static array sizes. This last part can drastically change the way one codes as everything needs to be pre-allocated, and no dynamic sizing can exist (e.g., when storing episodic trajectories, if they can have different lengths then the arrays to store them need to be oversized to fit the longest trajectory possible, and the unused parts are then filled with padding).

**Functional Programming** Thirdly, JAX is a purely functional programming language. While most modern programming languages are multi-paradigm and support functional tools as well as imperative tools, JAX enforces functional principles through its compilation process. Most notably, data immutability and pure functions can be a challenge when coming from an imperative language background.

**Control Flow** Fourthly, control flow (loops and conditions) is a bit different from normal Python. While strictly speaking those are consequences of the functional programming paradigm, they are important enough to be considered separately as they are the first issue one might encounter when moving their codebase from Python, or even non-jitted JAX, to jitted JAX. Indeed, Python's native `for` or `while` loops and `if` statements are not compatible with `jax.jit` and need to be replaced with JAX equivalents: `jax.lax.fori_loop`, `jax.lax.scan`, `jax.lax.while_loop`, and `jax.lax.cond` or `jax.lax.select`. Their heavier syntax, when compared to their Python counterparts, can be overwhelming at first. It is a consequence of the stateless aspect of functional programming, where in loops the evolution across iterations needs to be explicitly stored and fed to the next iteration.

Concerning conditions, as the values inside arrays are not known before execution but the code still needs to compile beforehand, two solutions exist. The first solution is if the condition depends on a "static" argument that stays constant throughout the whole program (e.g., a hyperparameter, like the type of activation layer in the neural network) and is thus known at compilation time. If the condition depends on a runtime variable (e.g., when the agent reaches a terminal state), then the JAX version of conditions (`jax.lax.cond` or `jax.lax.select`) must be used, as they will allow inspection of the actual value. However, both branches created by JAX conditions must be compiled (as the compiler cannot know which one will be taken

beforehand), and in the case of `jax.lax.select`, both are actually executed, which is inefficient and must be limited as much as possible to minimize runtime.

**Side-Effects and Logging** Fifthly, due to the pure aspect of functional programming, no side-effects are allowed by default, which prevents standard logging. To overcome this, side-effects must be explicitly flagged to allow data to "leak" out (using `jax.debug.print` or `jax.debug.callback`) of the pure program to allow for logging during execution, at the expense of additional runtime.

**Random Number Generation** Finally, but still very important to RL experiments, JAX enforces explicit random number generation (RNG) seeding. This explicit definition and passing is in contrast to some frameworks where the seed is defined once at the beginning of the program (which may be sensitive to order of operations or side effects) or worse, not defined at all. In JAX, every time you need to use the RNG you must feed an argument called a key, this key will not change unless you explicitly tell it to (using `jax.split`), and it needs to be explicitly passed to every function that will use it. A good practice is to have a different key for every random process (e.g., a key for network weight initialization, a key for action sampling, a key for environment step, etc.). This does not prevent the whole program being defined by a single seed; subkeys will be created by the primary key fixed by the seed. Having multiple separate RNGs that stem from the same seed allows independent reproducibility: if the RNG was shared, making changes to the actor and its sampling would impact the RNG for the environment. However, it would be best for those things to be independent to make debugging easier, and actually being able to tell if changes come from the randomness or some other unaccounted mechanism. This explicit key system is also a good way to see how randomness propagates through the algorithm/codebase.

Table 6.1 provides a side-by-side reference of the most common patterns in JAX and their Python counterparts, covering the differences discussed above: control flow constructs, in-place mutation, and side effects. It is intended as a practical guide for practitioners making the transition from imperative Python to functional JAX.

**The cost is worth it:** Most of these aspects are double-edged swords in disguise. While they seem like drawbacks at first, especially if coming from an imperative framework like PyTorch, they are in the end better practices to ensure the efficiency, the reproducibility[a], and the safety (due to strict typing, it will prevent some mixed-types operations that would have been unsafely allowed by Python's dynamic typing, preventing silent errors) of the code.

[a] At least at iso-hardware and environment on CPU; for GPU bitwise-reproducibility is always complex

**Table 6.1** – JAX Functional Patterns vs. Imperative Python

| Concept | Standard Python | JAX Equivalent | Usage Context |
|---|---|---|---|
| Fixed Loop | `for x in A:`<br>`for i in range(N)` | `for x in A:`<br>`for i in range(N)` | Only works for static variables (`A` or `N`). The loop is unrolled during tracing. |
| Indexed Loop | `for i in range(N)` | `jax.lax.fori_loop(`<br>`low, high, body, init)` | Efficient for loops where the index drives logic and state is minimal. |
| Scanned Loop | `for x in A:`<br>`s = f(s, x)` | `jax.lax.scan(`<br>`f, init, A)` | The standard "for loop" in JAX. Explicitly handles state across iterations and collects results. |
| Selection | `val = x if p else y` | `jax.lax.select(p, x, y)` | **Masked Execution:** Both branches execute; faster for scalars/arrays by avoiding CPU/GPU branching. |
| Conditional | `if x > 0:` | `jax.lax.cond(p, f1, f2)` | **True Branching:** Only the chosen branch executes. Use for heavy computations. |
| Mutation | `x[i] = val` | `x.at[i].set(val)` | JAX arrays are immutable; this returns a new array with the update. |
| Debugging | `print(x)` | `jax.debug.print()`<br>`jax.debug.callback()` | Bypasses functional purity to "leak" data or side-effects for logging. |

### 6.2.4 Environments libraries in Jax

Environments written in Jax are in the end efficient environments that can run on CPU and GPU alike. The main libraries providing reference environments in Jax are:




**Gymnax** (Lange, 2022): The closest equivalent to gymnasium (Towers et al., 2024), offering a collection of classic control (Cartpole, pendulum) environments, the b-suite (Behaviour-Suite, Osband et al. (2019)), some MinAtar (Young and Tian, 2019) environments, and a few others.

https://github.com/RobertTLange/gymnax




**Brax** (Freeman et al., 2021): The MuJoCo (Todorov, Erez, and Tassa, 2012) equivalent. It includes most of the same environments. It is however being deprecated in favor of MuJoCo playground.

https://github.com/google/brax




**MuJoCo Playground** (Zakka et al., 2025): This is the equivalent of the Deepmind control suite (Tunyasuvunakool et al., 2020), containing the MuJoCo environments and more complex physics-based locomotion tasks. It is maintained by the authors of Brax at Google Deepmind.

https://github.com/google-deepmind/mujoco_playground



**OCTAX** (Radji, Michel, and Piteau, 2026): A collection of Chip-8 Games, very similar to the Atari (Marc G. Bellemare et al., 2013) environments but with a significant increase in speed.

https://github.com/riiswa/octax

While Jax offers incredible speed advantages, the smaller size of the community when compared to vanilla Python environments makes it so that some tasks may not yet be available as jax environments. Depending on the complexity of the missing environments, making them Jax compatible can be relatively easy. If they follow the gymnasium framework, switching to gymnax is a breeze. More recently, a method to automatically create efficient version of existing environments using LLM agents has been proposed by Karten, Appapogu, and Jin (2026).

### 6.2.5 Memory limitations of GPU computing

One limiting factor of using GPUs for parallelization is their limited memory (VRAM). Although modern GPUs have scaled their VRAM significantly to fit the larger and larger Large language Models (LLM), the high-end models that provide very large amounts of VRAM (80 to 94 GB for the H100) are expensive and may even be hard to get due to their high demand. Most standard GPUs from the past 10 years have between 11 and 32 GB of VRAM. By comparison, the standard RAM on dedicated computing servers would most likely be between 128 and 512 GB. As the goal of doing deep RL on GPUs is to parallelize as much as possible, this means fitting multiple times all the necessarily data on the GPU memory. On CPU, as the parallelization is limited to begin with, memory footprint is not an issue. However, fitting hundreds of parallel environments, or parallelizing multiple agents can quickly lead to overflowing the VRAM if their memory footprints are too big. One common occurrence of this issue is using agents with large replay buffers, and parallelizing lots of them. This requires to fit a large number of the replay buffer, which due to its large size (e.g., usually 1 million steps for SAC) quickly surpasses the available memory. This means that for environments with a large observation and action shape and complex dynamics, or for agents with large replay buffers, parallelization capabilities may be reduced. If one wishes to make the most out of parallelization on GPU, preferring memory-light agents would be beneficial. As the benefits of massive parallelization are significant as seen previously, we believe this constraint is justified.

**Industrial perspective**

As mentioned in the previous Section 6.1, for our industrial problem and similar ones, we believe that a very efficient parallel simulation is key to obtaining a policy robust to model misspecification, in realistic time. As a consequence, using Jax to implement these simulations and port them to GPU seems like the best technical choice to adopt.

**Frugal experiments** While in this chapter we advocated for more efficient implementations, including most specifically environments, we also want to emphasize that having access to more computing power and faster simulations should not automatically lead to conducting experiments on more complex environments. The experimental requirements to study algorithms and publish papers should not blindly scale with the ease of computation. For example, Kondo et al. (2020) showed that it was possible to experimentally study properties of the Rainbow algorithm (Hessel, Modayil, et al., 2018) using simple control environments (LunarLander, Cartpole, Acrobot, MoutainCar) with similar results to the much more complex and expensive Atari environments (Marc G. Bellemare et al., 2013). Ideally, environments should be picked not based on their complexity but on the characteristics of an algorithm they can test, a prime example of this philosophy being the Behavioral Suite for RL (bsuite, Osband et al. (2019)).

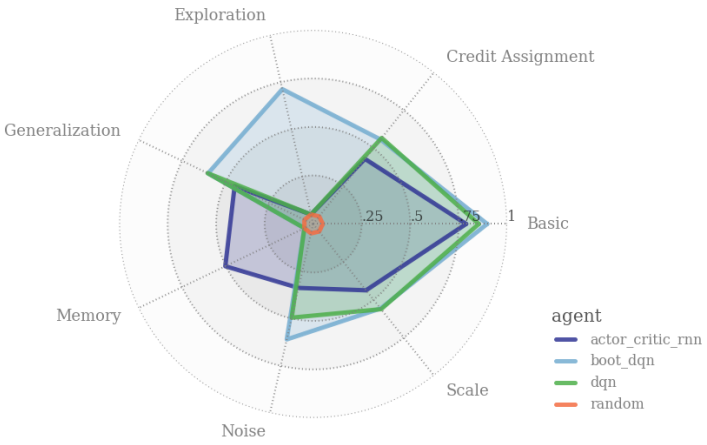



**Figure 6.6** – Algorithmic properties evaluated using `bsuite`, taken from Osband et al. (2019).

The lighter the experimental suite, the more accessible the research will be, and therefore the more insightful knowledge the community will be able to build.

## 6.3 Conclusion

This chapter examined parallelization in deep RL from two complementary angles: algorithmic design and practical implementation.

On the algorithmic side, the central insight is that parallelization is not merely an execution speedup: it reshapes the statistical properties of the training data. Collecting transitions from multiple independent environments simultaneously reduces temporal correlation and non-stationarity far more effectively than a single sequential agent could, even at equivalent sample counts. It is not the parallel actors themselves that matter in the abstract; it is the simultaneous exposure to decorrelated experiences that stabilizes gradient-based optimization. At scale, the diversity of environments also improves policy robustness and directly narrows the sim-to-real gap. The history of the field reflects a progressive shift from asynchronous CPU-based pipelines (A3C, IMPALA) toward synchronous GPU-resident approaches (Learning to Walk, PQN, FastTD3), with the recent finding that off-policy algorithms, long considered incompatible with the massively parallel regime, can match or exceed on-policy counterparts when properly configured.

On the implementation side, the key enabler of end-to-end GPU execution is JAX. Through its functional programming model and XLA compilation backend, it eliminates the PCIe transfer bottleneck that forces synchronization at every environment step, and makes it practical to run environments and agents entirely on GPU at the scale that massive parallelization requires.

From an industrial perspective, the findings of this chapter point toward a compelling alternative to sample-efficient methods that learn slowly and directly on the target system. Training a policy on a fast, coarse simulation with thousands of parallel environments produces agents that are robust to model misspecification precisely because of the enormous variety they have encountered. Fine-tuning on the real system then becomes a relatively short final step. This pipeline is not merely theoretical: it is already the standard approach in robotics, and the question has shifted from whether it works to how to engineer the simulation. The algorithms studied in the following chapters do not follow this route, as the computational and engineering overhead would have exceeded the scope of a single PhD. Nonetheless, the author considers it the most promising path for practitioners seeking robust real-world results in complex control settings, and would recommend it to anyone with the resources to pursue it.

Understanding the advantages of parallelization, both in raw speed and in its deeper algorithmic properties, is one thing. Translating that understanding into working code is another. The tools required to leverage massive parallelization in practice are not available off the shelf: building on top of JAX requires a functional programming mindset that is far removed from the imperative Python style most RL practitioners are trained on, and the broader ecosystem of parallelized deep RL frameworks remains sparse compared to its single-environment counterparts. Chapter 7 presents the software contributions developed during this thesis to bridge that gap, turning the theoretical case for parallelization made here into a concrete and usable research platform.

# Chapter 7

# Reliable and Efficient Implementations:

## Building the Tools for This Thesis

> Paying down technical debt may initially appear less glamorous than research results usually reported in academic ML conferences. But it is critical for long-term system health and enables algorithmic advances and other cutting-edge improvements.
>
> Machine Learning: The High-Interest Credit Card of Technical Debt, Sculley et al. (2014)

In the previous chapter we established the theoretical and practical case for parallelization in deep RL. Complementary to efficiency, reliability is a prerequisite for results to be interpretable: an efficient but incorrect implementation merely produces wrong results faster. This chapter presents the tools developed during this thesis to address both concerns. Section 7.1 surveys the main sources of unreliability in deep RL and introduces **ProbingEnvironments**, a plug-and-play integration test suite for actor-critic algorithms. Section 7.2 presents **Agents in JAX** (**AJAX**), a JAX-based deep RL library built around end-to-end GPU execution and seed-parallel training. Section 7.3 presents **TargetGym**, a JAX-native benchmark suite for setpoint-tracking control tasks. Together, these three contributions form the experimental platform developed during this thesis, on which the algorithmic work of Chapter 8 is built. All implementation work developed as part of this thesis is open-source.

### Contents


**7.1 Reliability in RL** . . . . . **106**
**7.2 AJAX** . . . . . **113**
**7.3 TargetGym** . . . . . **119**
**7.4 Conclusion** . . . . . **125**

## 7.1 Reliability in RL

Reliability in the context of deep RL implementations means having ways of ensuring that code matches the expected behavior described by the pseudocode. Lacking reliability leads to two distinct failure modes: reproducibility issues, where different implementations of the same pseudocode yield different results; and misinterpretation issues, where results are attributed to the algorithmic properties of the pseudocode rather than to what was actually executed. Pineau et al. (2021) propose a general checklist for reproducibility in Machine Learning (ML) papers; the following subsections focus on the failure modes we consider most harmful in practice.

### 7.1.1 Algorithm Pseudocode

A significant share of the work toward reliability concerns the pseudocode itself: if something is left ambiguous or hyperparameter values are left unspecified, the implementation cannot compensate. The potential issue this may cause is that the same pseudocode could lead to multiple implementations which have different results where we would expect similar if not identical results. Some algorithms and their pseudocode are deliberately left incomplete, for example when the method they describe is a generic method applicable to various algorithms. However, when there is no explicit justification for why the code should be left vague, there should be as little room as possible for interpretation in pseudocode. The irony of this approach is that defining unambiguous pseudocode is an ambiguous process itself. We believe the key principle is to make sure the implementation of an algorithm matches its pseudocode, and vice versa. As researchers would most likely go back and forth between the pseudocode and its implementation during experiments, one should make sure that the pseudocode still matches the implementation that produced the final results. If this requirement is not met, two versions of the algorithm exist: the one presented in the paper, and the one actually implemented. This can lead to reproducibility issues, where another researcher would try re-implementing the algorithm based on the pseudocode and would end up with potentially different results. PPO is a major example of this issue, where the performance increase between PPO and its predecessor TRPO comes, for the major part, from unreported[1] code-level optimizations, as claimed by Engstrom et al. (2020). DDPG also belongs to the same category, as implementing the pseudocode provided by the original paper did not lead to stable reproducibility of its result. Fujimoto, Hoof, and Meger (2018) claim that the actual core of the DDPG algorithm was, once again, unreported implementation details (e.g., Target network or clipped double Q-learning).

[1] in the pseudocode and the main body of the paper

### 7.1.2 Implementation Tricks

A major cause of the divergence between pseudocode and the actual implementation of the algorithm are "little" implementation details, or code-level optimizations, commonly referred to as implementation "tricks". Tricks are a collection of unreported hyperparameters, unreported design choices, and unreported reliance on default choices of used libraries. The most common culprits are:

- Neural network initialization: Both the type of initialization and the values it may use.
- Neural network optimizer: Both the optimizer itself and its hyperparameters.
- Normalization: Both for observations and rewards.

However, the full list of these unreported tricks can be much longer (e.g., Shengyi Huang et al. (2022) report 37 tricks for PPO, and Andrychowicz, Raichuk, et al. (2021) study the impact of more than 50 tricks).

While it may seem that these design choices are insignificant, and that algorithms sharing the same core ideas should yield similar results, the opposite can in fact be true. Engstrom et al. (2020) showed that most of the performance gain of PPO over TRPO comes down to these implementation tricks rather than from the algorithmic evolutions between the two algorithms. Moreover, they also state that these tricks fundamentally change how RL methods work. Andrychowicz, Raichuk, et al. (2021) did an even more exhaustive study, focusing solely on PPO, and investigating the impact of various implementation decisions on the performance of the agent, once again showing the importance of these often unreported design choices. One key insight from their work is the important impact of the initial action distribution on the performance.

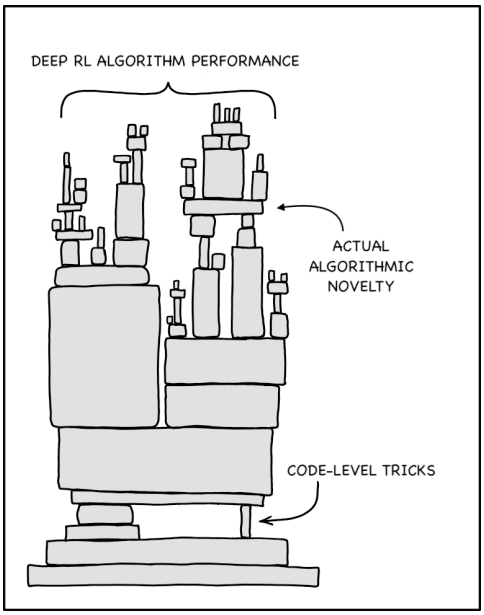



**Figure 7.1** – Representation of the impact of tricks on algorithm performance, adapted from Munroe (2020).

### 7.1.3 Unit tests in RL

In classic software engineering, reliability translates into testability. Tests are a collection of checks on the results and operations of software to make sure they match the expected behavior. There are two main categories of software tests:

**Integration tests** are tests that evaluate the whole software or major parts of it at once, aimed at testing the interactions between the different parts of the code.

**Unit tests** are tests focusing on small units of code. They verify that functions and objects behave as expected under most (ideally all) expected situations (potentially including invalid situations, where an error should be raised). While they cover smaller fractions of the code than integration tests, they are still vital as they allow one to pinpoint the origin of a failure.

Tests have two main benefits in the software engineering process:

- Assessing the functionality of newly written code. Sometimes handled through the Test Driven Development (TDD, Beck (2003)) framework.
- Ensuring that already existing code still works and has not been wrongly impacted by new functionalities. This is referred to as non-regression.

As deep RL algorithms can be complex from a code perspective, mistakes and bugs can quickly appear. They can be (1) annoying if they prevent the execution, making the development

process slower; or (2) dangerous if they do not prevent the execution but alter the results without being identified (silent errors). Using tests in deep RL should therefore be considerably beneficial toward the development process and ensuring that the code actually represents what the authors wanted to implement.

**Ground truth in RL** However, the key problem with testing in deep RL is the usual lack of ground-truth to compare the experimental results to. Indeed, unless someone tries to replicate results, they do not already know the results to expect. Even in the replication case, unless some specifics of the original implementation are strictly respected (e.g., using the same deep learning framework, the same libraries), results may vary slightly and introduce the need for approximate tests for which the uncertainty may be hard to set. While some people use some form of unit tests in RL, it is almost exclusively restricted to non-RL aspects of the software. In practice the "testing" is most of the time limited to observing the performance and some metrics of the resulting algorithm and subjectively assessing their truthfulness. As for the unit tests, most small functions of RL codebases can be tested to their full extent using dummy data, and are therefore not subject to the same issues as the larger task-dependent integration tests.

**Formal verification** A more rigorous approach to software correctness is formal verification, which consists in mathematically proving that a program satisfies a given specification. Tools such as Coq (The Coq Development Team, 2024), Why3 (Filliâtre and Paskevich, 2013), or Frama-C (Cuoq et al., 2015) allow one to derive machine-checked proofs that a piece of code matches its intended behavior. This approach has had major successes in safety-critical software (compilers, operating system kernels, cryptographic protocols). However, formal verification is not applicable to deep RL implementations in the current state of the art. Deep RL algorithms rely on floating-point neural networks trained by stochastic gradient descent on a non-stationary objective: their behavior is inherently probabilistic, their convergence properties are not amenable to closed-form specification, and the "correctness" of an implementation cannot be reduced to a deterministic functional contract. Formal methods verify logical properties of programs; they have no framework for specifying that a policy should converge to near-optimal behavior in expectation over random seeds. This leaves testing and empirical validation as the only practical tools for reliability in deep RL.

**Probe Environments** Indeed, as the expected performance of a given algorithm on specific tasks is unknown a priori, we cannot provide meaningful targets for integration tests (unless making risky assumptions with large uncertainty, leading to ultimately mostly meaningless tests). However, people in the community have hypothesized that tests could still exist in RL using Probe Environments (Jones, 2021). The key insight is that while expected performance on complex tasks is unknown a priori, we can design very simple environments (think 2-state MDPs) for which we can easily compute the optimal behaviors (e.g., optimal policy return

to evaluate the policy, optimal value functions to evaluate the critic), allowing us to test the algorithms with high precision on their expected behavior. Of course, this approach has a few caveats. First, it expects the algorithm to converge to or near the optimal solution, which may be hard to guarantee under a reduced budget (as we wish the unit tests to be as quick as possible for practical use). Second, the simplicity of these tasks does not guarantee that the algorithm will succeed on more complex tasks. However, while passing these tests is not predictive of the performance on more complex tasks, we believe that failing these tests is a good indicator of a problem in the code. Moreover, Jones (2021) argued that it should be possible to pinpoint the origin of the problem (e.g., actor network, critic network, advantage computation, etc.) through a series of simple tests. As for the convergence issue, there is no clear solution; however, it does point to the sample (in-)efficiency of deep RL algorithms on very simple tasks under their standard hyperparameters. The idea of Probe Environments is therefore not perfect, but it is most likely the best proxy for integration tests we can have.

While the idea of Probe Environments is sound, only the core concept was provided by Jones (2021), leaving practitioners without a ready-to-use implementation. We have addressed this gap with **ProbingEnvironments**, a Python library providing a plug-and-play suite of such tests for deep RL actor-critic algorithms.

The package can be found at: https://github.com/YannBerthelot/ProbingEnvironments

Since implementation following software engineering's good practices is a core tenet of this thesis, building this library was a natural contribution. The library implements six tests, each targeting a specific component of the algorithm:

- **Value network loss**: verifies that the critic's optimizer actually minimizes the loss and that the network weights are updated.
- **Value network backpropagation**: checks that gradients flow correctly through the critic, catching issues such as accidentally detached tensors or frozen layers.
- **Reward discounting**: ensures that the discount factor $\gamma$ is applied correctly in the return computation.
- **Advantage-policy alignment**: verifies that the advantage estimates correctly guide the actor update.
- **Actor-critic coupling**: checks that the policy actually improves in response to the critic's value estimates.
- **Batch processing**: validates that the algorithm behaves correctly when processing batches of transitions.

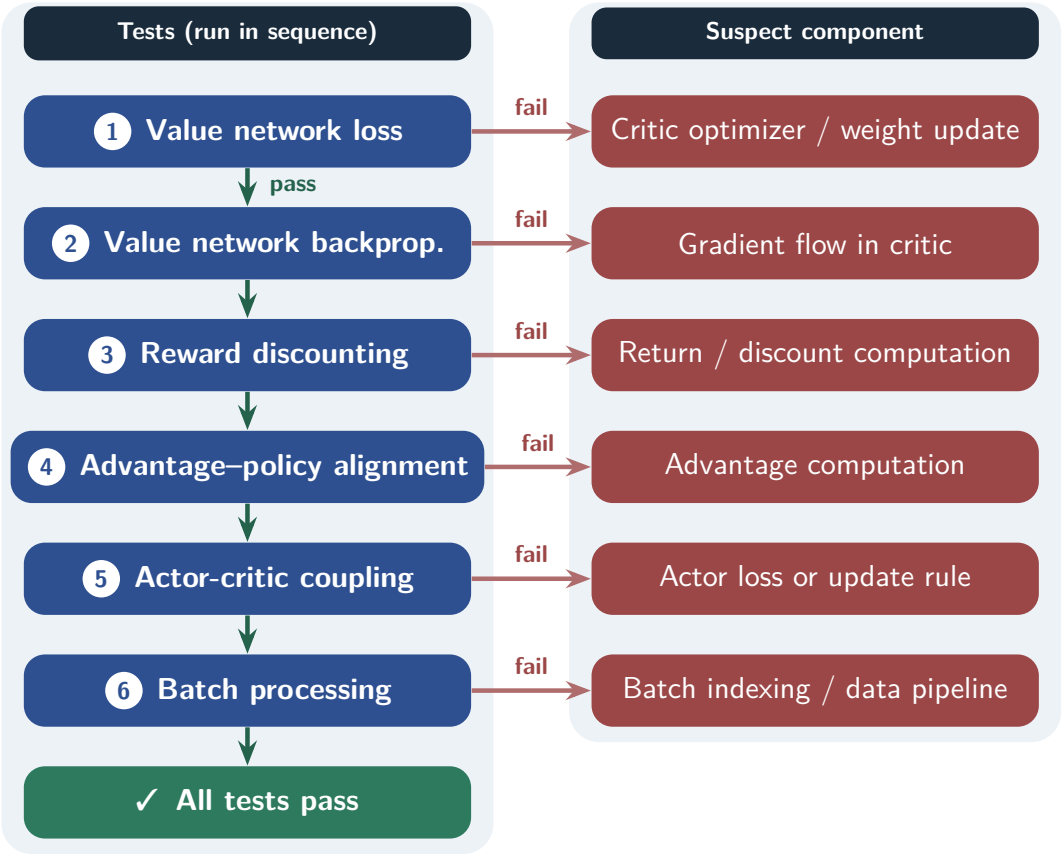



**Figure 7.2** – Diagnostic flowchart of the *ProbingEnvironments* test suite. Tests are run in sequence; a failure immediately identifies the suspect algorithmic component without requiring the practitioner to inspect the full codebase.

The sequential structure of these tests is key to their diagnostic value. Consider a practitioner who mistakenly applies no discount when computing the returns (i.e., $\gamma = 0$). Tests 1 and 2 would pass (the critic can still learn to minimize a loss, and gradients flow correctly), but test 3 would fail, immediately identifying the return computation as the source of the problem. Without such a suite, the same bug would typically only manifest as degraded performance on a complex benchmark, offering no indication of where to look. This systematic narrowing is precisely what is otherwise missing from the "observe the training curve and hope" approach that currently dominates RL debugging practice.

A central challenge in such a library is bridging the variability of Application Programming Interface (API) designs across RL codebases. **ProbingEnvironments** addresses this through a framework adapter system: a ready-to-use adapter for SB3 (Raffin et al., 2021) is provided without additional configuration, while a template adapter allows users to connect any custom implementation with minimal effort. The library is installable directly via `pip` from its repository.

Additional tests and adapters are ongoing work, as broadening coverage across algorithmic components and frameworks will further increase the library's diagnostic value.

In the interest of transparency, ProbingEnvironments was not used within this thesis itself. Writing framework adapters for both the SB3-based EVarEst codebase and the later AJAX library would have required non-trivial integration effort that competing priorities did not allow. This is, in a sense, its own finding: even the author of such a library found adoption within their own projects impractical, which speaks to the broader difficulty of integrating testing infrastructure into RL codebases whose API designs vary widely. The contribution stands as a proof of concept and a reusable tool for the community, packaged to minimize the integration barrier for future users, and its further development remains planned work.

### 7.1.4 Hyperparameter Sensitivity

Hyperparameter sensitivity is a reliability concern as much as a performance one. When experimental results depend on hyperparameter choices that are unreported or selected without systematic search, it becomes impossible to attribute a performance gain to an algorithmic contribution rather than to a favorable configuration. This confounding is one of the most well-documented sources of irreproducibility in deep RL (Islam et al., 2017; Henderson et al., 2018; Pineau et al., 2021): two implementations of the same algorithm, trained with different hyperparameters, can yield results divergent enough to make comparisons between papers effectively meaningless (Agarwal et al., 2021).

The brittleness of algorithms is therefore worth studying explicitly, as not all hyperparameters are equally dangerous. Henderson et al. (2018) studied the impact of various hyperparameters on the performance of PPO, TRPO, and DDPG, and showed how specific changes that would a priori seem harmless could in practice lead to significant degradation in policy performance. For example, a large policy network (400, 300) prevented learning on HalfCheetah, while using a tanh activation for the policy network led to TRPO underperforming when compared to ReLU (V. Nair and Hinton, 2010) or Leaky ReLU (Maas, Hannun, and Ng, 2013) on the same environment. More broadly, Andrychowicz, Raichuk, et al. (2021) showed that architectural choices such as network size, activation function, and initialization can completely prevent learning, while others such as the entropy coefficient or the target update rate tend to be more forgiving. Identifying which hyperparameters lie in the dangerous category is arguably as important as tuning their values.

There is no universal solution to this problem, but two complementary approaches can limit its impact. First, systematic hyperparameter search reduces the risk of reporting results that depend on a fortunate configuration; in Chapter 8 we use Optuna (Akiba et al., 2019) for this purpose. Second, and more fundamentally, algorithms can be designed with robustness to their own hyperparameters as an explicit goal. SAC (Haarnoja, A. Zhou, Abbeel, et al., 2018) is a

prominent example: its maximum entropy framework prevents premature policy collapse and yields more stable training, which the authors explicitly contrast with the brittleness of prior off-policy methods such as DDPG. A follow-up formulation (Haarnoja, A. Zhou, Hartikainen, et al., 2019) further reduces sensitivity by automatically tuning the entropy temperature, removing one of the most consequential manual decisions. Beyond individual algorithms, there is also a cost argument: each additional hyperparameter multiplies the search budget, and the cumulative computational cost of large-scale sweeps is non-trivial in terms of energy and carbon footprint (Schwartz et al., 2020). When sensitivity cannot be eliminated, it should at minimum be documented and characterized through ablation studies.

These reliability concerns (unreported pseudocode, implementation tricks, and hyperparameter sensitivity) were the direct motivation for the toolchain built during this thesis. The following sections describe AJAX and TargetGym: the implementation library and the benchmark environments developed to conduct the experimental work of this thesis under the reliability and efficiency standards outlined above.

## 7.2 AJAX

### 7.2.1 Motivation

Just as the design and assembly of an experiment in physics can take a vast portion of a PhD, likewise the implementation and experiments with deep RL algorithms we did during this thesis represented a large part of our work. Indeed, while we believe in theoretically grounded ideas, theoretical guarantees can, however, be limited in deep RL when compared to tabular RL setting or even bandits. Therefore, since experiments are key to assessing the validity of claims and ideas in deep RL, we wanted to use tools suited for the job. Tools allowing for reproducibility, ease of modification, and efficiency.

We began the experimental section of this thesis with the EVarEst study (see Chapter 5). In this study we used a customized fork of the SB3 library. This choice was motivated first by the economy of using existing software, and second by the popularity of this library and its perceived reliability. However, in hindsight, this choice proved to be a mistake. First of all, SB3 is implemented using the PyTorch (Paszke et al., 2019) deep learning library. When compared to JAX performance, this is relatively slow, and for our experiments was limited to CPU-only experiments (as the CPU-GPU bottleneck would have been too significant to consider a hybrid setup, and end-to-end GPU execution is not a feature of SB3). Second, while SB3 is very simple to use and offers a large variety of agents, we found it quite hard to understand and modify, in part due to the modularity required by this number of different agents. As a consequence, the experimental phase of EVarEst was quite painful, requiring massive distributed computations on hundreds of CPU cores on the Grid5000 (g5k, Balouek et al. (2013)) servers, and taking

weeks on the MuJoCo experimental suite. This experience, as well as the will to try out JAX, led us to implement our own agents, in the AJAX library.

### 7.2.2 AJAX

AJAX (Berthelot, 2025a) is a library providing implementations of deep RL algorithms written in JAX, designed around end-to-end GPU compatibility, efficiency, and ease of modification. The choice of algorithms to implement was directly driven by our experimental needs, and most notably by the desire to experiment with average-reward RL algorithms as part of early exploratory work. Since average-reward algorithms are absent from popular libraries such as SB3, CleanRL ( Shenanyi Huang et al. (2022)), MushroomRL ( D'Eramo et al. (2020)), and Ray Rllib ( Liang et al. (2018)), and since supporting them may require rethinking some foundational module structures, this gap further reinforced the decision to build our own package.

The package can be found at: https://github.com/YannBerthelot/Ajax

The library currently provides the following algorithms, organized around pairs of discounted-reward and average-reward variants, complemented by two additional off-policy contributions:

- **PPO and Average Policy Optimization (APO, X. Ma et al. (2021))**: an on-policy policy gradient algorithm and its average-reward counterpart.
- **SAC and Average-reward Soft Actor-Critic (ASAC, Adamczyk et al. (2025))**: an off-policy actor-critic algorithm and its average-reward counterpart.
- **REDQ**: a randomized-ensemble off-policy algorithm for high UTD.
- **Action Value Gradient (AVG)**: an incremental average-reward version of SAC for low-memory footprint.

**Design philosophy** AJAX is built around the functional programming model imposed by JAX. Agents are implemented as stateless, pure functions: all mutable state (network parameters, optimizer state, replay buffers) is carried explicitly as data rather than stored inside objects. While this is a significant departure from the object-oriented style of PyTorch-based libraries, it is what enables JAX's core transformations, namely just-in-time compilation (`jit`), automatic vectorization (`vmap`), and automatic differentiation (`grad`), to apply seamlessly to the full training loop. The result is a codebase that is more amenable to modification at the algorithmic level, since there are fewer layers of abstraction to navigate, at the cost of a steeper initial learning curve. Memory efficiency is also handled at this level: by passing replay buffers as donated arguments to Just In Time (JIT)-compiled functions (`donate_argnums`), JAX can

update them in place on the GPU without duplicating their contents, which matters when buffers contain millions of transitions.

**Environment compatibility** A deliberate consequence of the end-to-end JAX design is the choice of compatible environment backends. AJAX supports Gymnax (Lange, 2022), a JAX-native suite of classic control and MinAtar environments, and Brax (Freeman et al., 2021), a JAX-based physics simulator providing continuous control benchmarks comparable to MuJoCo. Because both backends are written entirely in JAX, environment steps can be fused into the same compiled computation graph as the agent update, keeping the full rollout-and-train loop on the GPU. Gymnasium (Towers et al., 2024), the standard Python environment interface, is explicitly not supported by design: its CPU-bound Python step function would reintroduce the host-device transfer bottleneck that the JAX backend eliminates, and it cannot be vectorized over seeds. Supporting it would therefore undermine the two core properties (speed and seed parallelism) that motivate the library's architecture.

**Speed and seed parallelism** The end-to-end JAX design has a direct consequence on training speed. Because the full update step is compiled and executed on the GPU without ever returning to the host, the CPU-GPU data transfer bottleneck that affects PyTorch-based libraries is eliminated (see Figure 6.3 in Chapter 6 for a detailed illustration of this bottleneck). Beyond raw speed, JAX's `vmap` transformation makes it straightforward to vectorize the training loop over multiple seeds simultaneously: running $N$ seeds in parallel on a single GPU incurs only a small overhead compared to running a single seed, rather than the $N$-fold cost of sequential execution. This is directly relevant to statistical robustness: whereas the EVarEst experiments were limited to a handful of seeds due to computational constraints, AJAX makes it practical to routinely run tens of seeds per experiment, yielding tighter confidence intervals and more reliable comparisons. Figure 7.3 and 7.4 report wall-clock comparisons between AJAX (with Brax) and SB3 (with Gymnasium) on the Ant continuous-control task for SAC and PPO respectively, run on a single Nvidia L40S GPU, to illustrate this gain concretely.[2]

[2]While Brax and MuJoCo both implement the Ant task, they differ in their physical modeling and integration scheme. This comparison measures the speed benefit of the end-to-end JAX stack on an equivalent task, not an identical one; the two simulators should not be treated as interchangeable.

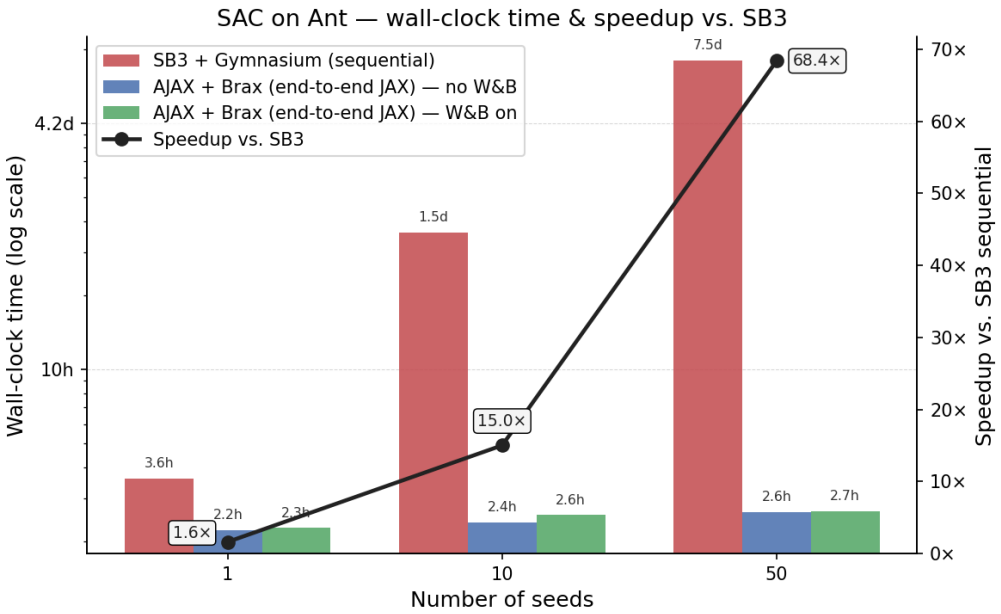



**Figure 7.3** – Wall-clock time and speedup of AJAX + Brax (end-to-end JAX) versus SB3 + Gymnasium (sequential) for SAC on the Ant environment, as a function of the number of seeds. The left axis (log scale) shows wall-clock time in hours; the right axis shows the speedup factor relative to SB3 sequential. Running with W&B logging incurs a modest overhead but does not change the scaling behavior.

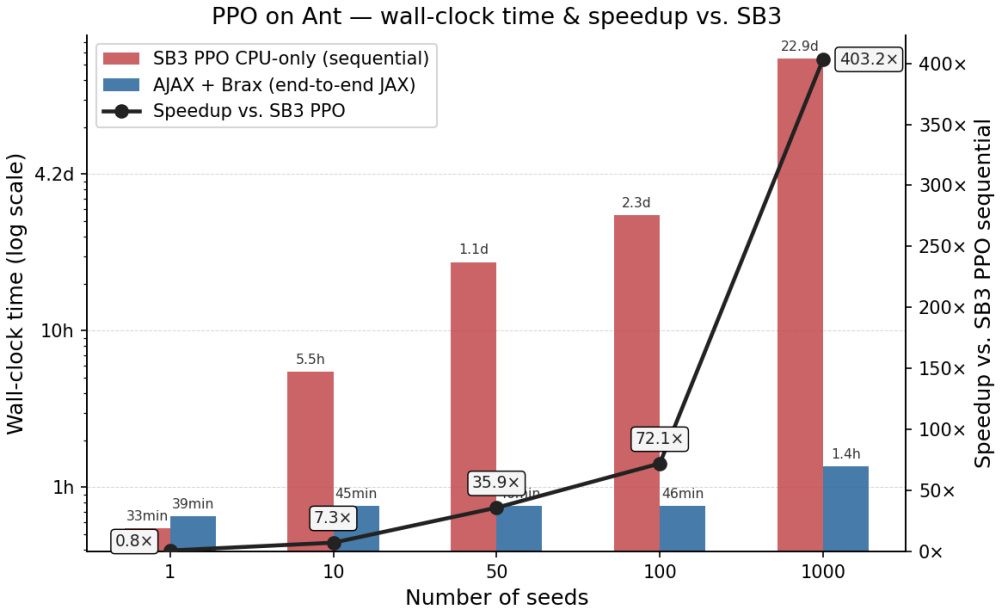



**Figure 7.4** – Wall-clock time and speedup of AJAX + Brax (end-to-end JAX) versus SB3 + Gymnasium (sequential) for PPO on the Ant environment, as a function of the number of seeds. The left axis (log scale) shows wall-clock time in hours; the right axis shows the speedup factor relative to SB3 sequential. W&B logging was not ablated here, as the SAC comparison already demonstrated its negligible overhead.

It is important to note that SB3 ran end-to-end on CPU, as it does not support end-to-end GPU execution, while AJAX ran end-to-end on GPU; the comparison is therefore between the two stacks as they would realistically be used in practice. From the SAC comparison (Figure 7.3), we can observe that, even for a single seed, the end-to-end JAX approach already speeds up computations by ∼60%, most likely thanks to JAX's compiled and more optimized nature. Then, while SB3 + Gymnasium's computation time scales linearly with seed count (all computation is sequential, as we assume everything is on a single device), AJAX + Brax shows highly sub-linear scaling: the speedup reaches 15× at 10 seeds, and at 50 seeds, what takes over 100 hours with SB3 takes roughly 2 hours with AJAX (a 68× speedup). While this speedup may be coming at a cost (somewhat increased code complexity, GPU VRAM limitations: 100 seeds for SAC on Brax Ant would not fit into the 46 GB memory of our GPU), it is undeniably significant. We believe the time spent developing this stack was worth every second, as this time was won back down the line in faster experiments.

The picture is somewhat different for PPO (Figure 7.4). Unlike SAC, at a single seed AJAX is slightly *slower* than SB3: PPO is an on-policy algorithm whose per-update computation is comparatively lightweight, so the overhead of JIT compilation and the end-to-end GPU pipeline is not fully amortized at low parallelism. The seed-scaling advantage is also more modest than for SAC: because on-policy updates are less compute-intensive, there is less GPU work to parallelize, which reduces the relative gain of `vmap`. That said, the speedup is still substantial at higher seed counts, confirming that the end-to-end JAX stack delivers meaningful acceleration for PPO as well, just with a higher break-even point in terms of number of seeds. Crucially, PPO's much smaller memory footprint, owing to the absence of a replay buffer (which for SAC stores 1 M transitions directly on the GPU), means that PPO can be scaled to far larger numbers of parallel seeds before hitting GPU VRAM limits, partially compensating for its lower per-seed speedup.

For now, this stack does not allow multiple hyperparameter values to be run in parallel, only parallel seeds. The core implication is nonetheless striking: speed translates directly into seeds, and seeds translate directly into statistical robustness. What was previously a budget decision (how many seeds can we afford?) becomes a near-free one. This means that even early exploratory runs, where one would traditionally use fewer seeds to save time, can now be conducted with the full seed count at no extra cost, delivering statistically reliable data from the very first experiment rather than only at the end of the research cycle.

While not studied in this thesis, we also note that this stack allows one to easily parallelize a large number of environments (as in Rudin et al. (2022)), or run with a large number of actor and critic networks. Most notably, AJAX already enables one to change the number of critics with a single hyperparameter. We also investigated leveraging these capabilities to build a massively parallel version of AVG, as its lightweight memory footprint allows fitting a

large number of agents onto a single GPU, but obtained no significant results within the time dedicated to this direction.

**Logging under end-to-end GPU execution** A practical challenge that arises with end-to-end JAX training is that the host process has no visibility into intermediate computation: since the full training loop runs as a single compiled kernel, standard logging calls cannot simply be inserted into the loop body. AJAX addresses this through JAX's `io_callback` mechanism, which allows host-side side effects (such as sending metrics to Weights & Biases (W&B, Biewald (2020))) to be injected at specific points within a compiled function without breaking the compilation. This enables the same mid-run monitoring that practitioners expect from conventional libraries, without sacrificing the performance benefits of end-to-end GPU execution.

**Configurability** A recurring need in our experiments was the ability to scale specific algorithmic components, such as the number of critics, the number of critic updates per environment step, or the network architecture, without rewriting the training loop. In AJAX, these are first-class configuration parameters, making it straightforward to sweep over them systematically. This was particularly useful for REDQ and ASAC, both of which are sensitive to these choices.

**Upcoming work** Native integration with Optuna (Akiba et al., 2019) for automated hyperparameter search is an ongoing development. While Optuna was used in the experiments of Chapter 8, it was not yet integrated directly into AJAX at that point, and its inclusion remains planned future work.

**Unit testing** Beyond its design principles, reliability in AJAX is actively enforced through a test suite of approximately 150 tests covering every implemented algorithm. Individual components are tested in isolation: buffer operations, network architectures, and algorithm-specific computations such as GAE, TD-error scaling, and action distribution log-probabilities. Each agent's full training pipeline is additionally exercised end-to-end: losses are verified to be finite, parameter updates are confirmed to change network weights in the expected direction, and training loops are validated across both Gymnax and Brax backends. Tests are parameterized over environments and configurations and run automatically on every commit, providing the non-regression guarantees that are otherwise absent from most RL codebases.

**Code quality and continuous integration** Alongside the test suite, code quality in AJAX is enforced through a combination of pre-commit hooks and a Continuous Integration (CI) pipeline. Before any commit is recorded, Ruff (Marsh and Software, 2023) enforces consistent formatting, import ordering, and a configurable set of linting rules, while Mypy (Lehtosalo et al., 2012) performs static type checking to catch type errors early. On every push and pull request to the main branch, a GitHub Actions pipeline reinstalls all dependencies via Poetry (Eustace

et al., 2018), reruns the full pre-commit suite, and executes the test suite with branch coverage measurement, rejecting builds that fall below a 70% coverage threshold; current coverage stands at 80%. Dependency management is handled by Poetry throughout: dependencies are pinned precisely, optional groups such as logging, profiling, and hyperparameter search are kept separate from the core install, and the package is published to PyPI, making it installable with a single `pip install` command.

**A learning experience** Beyond its role as an experimental tool, building AJAX was also a significant learning exercise. Implementing deep RL algorithms from scratch in JAX required engaging seriously with functional programming, with the constraints and idioms of JAX's programming model, and with the broader ecosystem of tools built around it (Flax (Heek et al., 2020), Optax (Hessel, Budden, et al., 2020)). While JAX is genuinely powerful and has become our tool of choice for experimentation, the path to fluency was non-trivial, and we believe that documenting this as part of the thesis work gives an honest picture of the practical cost of adopting these tools.

## 7.3 TargetGym

### 7.3.1 Motivation

Just as the lack of average-reward implementations motivated building AJAX, the lack of suitable benchmark environments motivated **TargetGym**. Standard RL benchmark suites are primarily designed for episodic, discounted-reward settings focused on reaching a goal over a finite horizon. However, a large class of practically relevant control problems does not fit this mold. Consider keeping an aircraft at a target altitude, maintaining a vehicle at a desired speed, or stabilizing a bicycle: these are continuing tasks where the agent must reach a target state and **remain** at it indefinitely. Such tasks are naturally framed as **Target MDPs** (Berthelot, 2025b), where the reward depends on proximity to a target set and the objective is to maintain that proximity over time rather than to reach it once.

Existing libraries offer few environments of this kind that are both physically grounded and computationally efficient. We therefore built **TargetGym**, a JAX-native suite of physics-based environments specifically designed for Target MDP research, with first-class support for parallelization and compatibility with both AJAX and SB3.

### 7.3.2 TargetGym

**TargetGym** (Berthelot, 2025b) provides a collection of physically realistic, JAX-native environments for RL on Target MDPs.

The package can be found at: https://github.com/YannBerthelot/TargetGym

**Design philosophy** All environments in TargetGym implement the Gymnax (Lange, 2022) interface, inheriting its purely functional design: states and parameters are represented as immutable Flax (Heek et al., 2020) dataclasses, making the full step function JIT-compilable and directly vectorizable with `vmap`. Physics is modeled from first principles rather than via lookup tables or simplified heuristics: aerodynamic forces, engine thrust curves, electric motor dynamics, bicycle mechanics, and chemical reaction kinetics are all derived analytically. The environments deliberately model challenges that make control difficult in practice, including actuator lags, momentum and inertia effects, altitude- and speed-dependent dynamics, and irrecoverable failure states. Numerical integration uses a fourth-order Runge-Kutta scheme (RK4, Hairer, Nørsett, and Wanner (1993)) with configurable sub-steps, balancing physical accuracy with computational efficiency.

A key conceptual distinction structures the suite. **Stable-Target MDPs** are tasks where the target is a stable equilibrium that the policy must regulate toward and maintain: perturbations dissipate naturally, and the challenge lies in reacting accurately to disturbances and setpoint changes (e.g., altitude tracking, speed control). **Unstable-Target MDPs** are tasks where the target is an inherently unstable equilibrium requiring continuous active control to prevent divergence (e.g., bicycle balancing). Without corrective action, the system departs from the target, making these tasks structurally harder and closer to classical active stabilization problems.

**Speed and parallelization** Because all environments are implemented entirely in JAX, the full rollout loop executes end-to-end on a GPU or Tensor Processing Unit (TPU). Time-step loops are compiled via `jax.lax.scan`, which eliminates Python-level loop overhead and reduces the rollout to a single GPU kernel. Parallel environment instances are obtained with minimal additional implementation cost via `jax.vmap`. Table 7.1 reports the throughput of each environment on CPU; for a concrete sense of the gains achievable with end-to-end GPU execution, the wall-clock comparison between AJAX (end-to-end GPU) and SB3 (end-to-end CPU) in Figure 7.3 illustrates the magnitude of this advantage.

**Table 7.1** – TargetGym environment throughput on CPU (steps/second, averaged over $10^8$ steps after JIT warmup).

| Environment | Type | Steps/s |
|---|---|---|
| Plane | Stable-Target MDP | ∼540,000 |
| Car | Stable-Target MDP | ∼850,000 |
| Continuous Stirred-Tank Reactor (CSTR) | Stable-Target MDP | ∼1,490,000 |
| Bike | Unstable-Target MDP | ∼1,770,000 |

**Compatibility** TargetGym environments implement the Gymnax interface natively, making them directly usable with AJAX without any adaptation layer. For practitioners using CPU-based libraries, a Gymnasium (Towers et al., 2024) wrapper is also provided for each environment, enabling use with SB3. It is worth noting that the Gymnasium wrapper incurs the same CPU-bound overhead as any standard Gymnasium environment: it is provided as a practical on-ramp for users not yet on the JAX stack, but it is not suitable for the end-to-end GPU execution that AJAX enables.

### 7.3.3 Environments

TargetGym currently provides four environments spanning three physical domains and one classic balance task, covering both types of Target MDPs. Each exposes a continuous observation and action space. Figure 7.5 shows a snapshot of each environment's visual rendering; Figure 7.6 shows example trajectories.

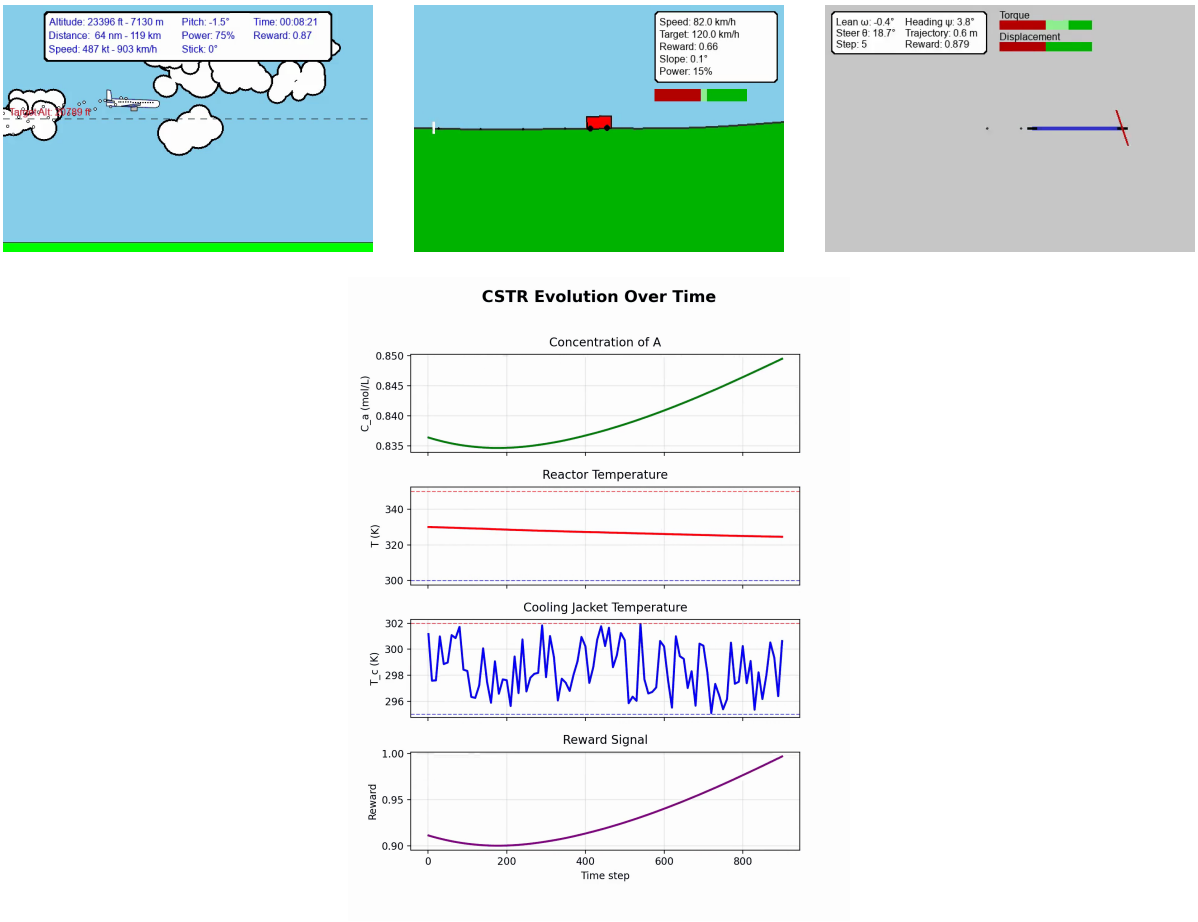



**Figure 7.5** – Visual renderings of the four TargetGym environments. Top row, left to right: Plane (altitude tracking), Car (speed tracking with road gradient), Bike (upright balancing). Bottom: CSTR (concentration regulation over time).

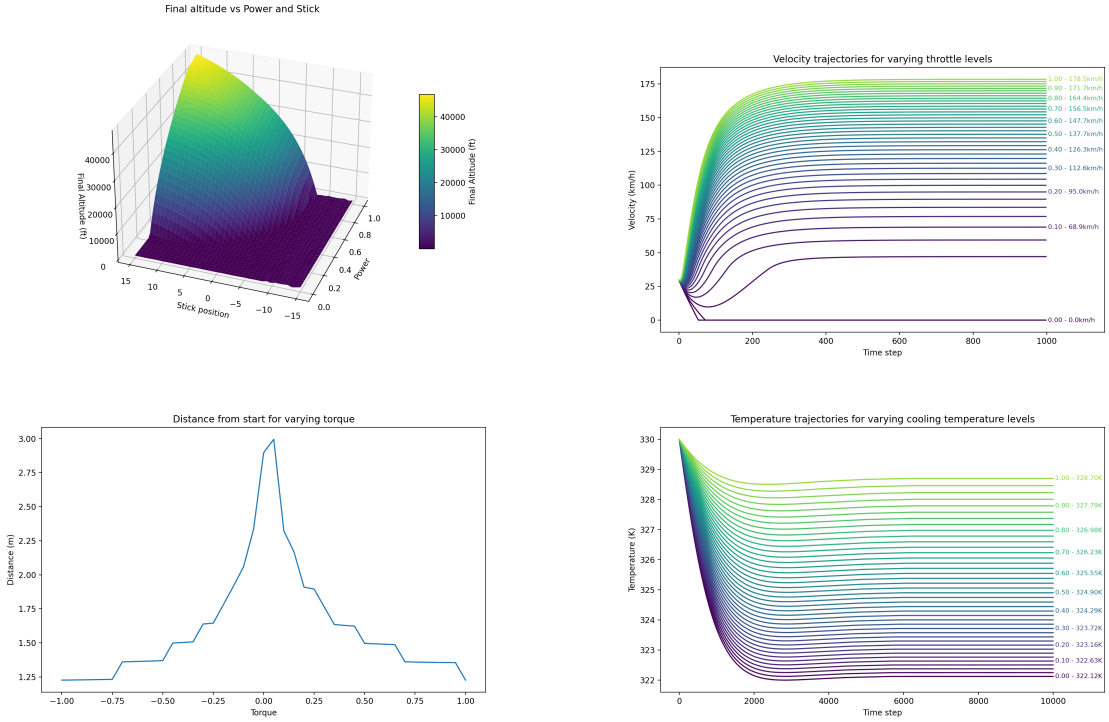



**Figure 7.6** – Example trajectories from the four TargetGym environments. Top left: Plane (altitude tracking). Top right: Car (speed tracking with road gradient). Bottom left: Bike (upright balancing). Bottom right: CSTR (concentration regulation).

**Plane** The Plane environment simulates the longitudinal flight dynamics of an Airbus A320-like aircraft in a 2D vertical plane. The agent controls throttle and elevator deflection to maintain a target altitude. The physics model includes nonlinear aerodynamics as a function of angle of attack and Mach number (including stall and compressibility effects), altitude-dependent air density via the International Standard Atmosphere, a full thrust model with altitude and Mach corrections, and first-order actuator lags for both throttle and elevator. The full derivation and parameter justification are provided in Appendix B.

- **Observation** (9-dim): horizontal speed, altitude, vertical speed, pitch angle, pitch rate, flight path angle, target altitude, current throttle setting, current elevator deflection.
- **Action** (2-dim): commanded throttle and commanded elevator angle, both in $[-1, 1]$.
- **Reward**: normalized altitude error raised to the 10th power, providing a shaped gradient near the target; large negative reward on ground collision or altitude limit breach.
- **Termination**: ground collision ($z \leq 0$) or maximum altitude exceeded ($z \geq 12{,}192$ m), or truncation after 10,000 steps.

**Car** The Car environment simulates an electric vehicle following a target speed on a road with varying gradient. The agent controls a combined throttle/brake input to track the setpoint. The physics model includes an electric motor torque curve (constant-torque at low RPM, constant-power at high RPM), drivetrain dynamics, aerodynamic drag, rolling resistance, and road gradient effects. A distinctive feature is the inclusion of a forward-looking array of road gradient sensors in the observation, making this a task requiring anticipatory control rather than pure reactive regulation.

- **Observation** (12-dim): current velocity, target velocity, 10 road gradient sensors over a 100 m look-ahead window.
- **Action** (1-dim): combined throttle/brake command in $[-1, 1]$.
- **Reward**: normalized velocity error raised to the 4th power; large negative reward on terminal states.
- **Termination**: velocity drops to 0 or exceeds $\sim$200 km/h, or truncation after 1,000 steps.

**Bike** The Bike environment implements the Randløv & Alstrøm bicycle balancing task (Randløv and Alstrøm, 1998), the only Unstable-Target MDP in the current suite. The upright equilibrium is inherently unstable, and the agent must apply continuous corrective torque and handlebar displacement to prevent falling. An optional navigation mode extends the task by requiring the agent to simultaneously balance and steer toward a target position, introducing a secondary objective that competes with the stabilization goal.

- **Observation** (5-dim): tilt angle, tilt angular velocity, tilt angular acceleration, steering angle, steering angular velocity.
- **Action** (2-dim): torque and handlebar displacement, both in $[-1, 1]$.
- **Reward**: normalized tilt error raised to the 4th power (dense mode), or a goal-proximity sparse reward (navigation mode); large negative reward on fall.
- **Termination**: tilt angle exceeds 12°, or truncation.

**CSTR** The CSTR environment simulates a Continuous Stirred-Tank Reactor, adapted from the PC-Gym suite (Bloor, Torraca, et al., 2026). The agent controls the reactor cooling temperature to regulate the concentration of reactant A at a target setpoint. The dynamics are governed by Arrhenius kinetics and integrated with RK4, yielding a nonlinear temperature-concentration coupling that is sensitive to both over- and under-cooling.

- **Observation** (3-dim): reactant A concentration, reactor temperature, target concentration.
- **Action** (1-dim): commanded cooling temperature in $[-1, 1]$, mapped to $[295, 302]$ K.

- **Reward**: normalized concentration error squared.
- **Termination**: temperature outside $[300, 350]$ K or concentration outside $[0.7, 1.0]$ mol/L, or truncation after 1,000 steps.

### 7.3.4 Upcoming Work

TargetGym is an actively developed library. Several extensions are planned on the environment side: additional PC-Gym tasks will be integrated, covering a broader range of process control problems; perturbations such as wind and turbulence for the Plane environment, uneven terrain for the Car, and sensor noise for the CSTR are under development, which will introduce stochastic elements and make the tasks more representative of real-world conditions. A more convenient interface for partially observable variants of each environment is also planned.

**Industrial perspective**

The TargetGym environments share the same setpoint-tracking structure as the industrial furnace problem that motivates this thesis: an agent must regulate a controlled variable to a time-varying target under nonlinear dynamics, with limited actuation authority and a shaped dense reward. This structural similarity makes TargetGym a natural proxy for evaluating algorithms before deploying them on the real system. Looking further ahead, we believe TargetGym could grow into a broader benchmark for industrial process control in RL: adding more tasks from the PC-Gym suite and beyond would cover a wider range of industrial dynamics, and a fast but deliberately coarse approximation of the furnace itself could serve as a direct test bed for the massively parallel training route advocated in Chapter 6. Such a rough simulation, run across thousands of parallel instances, could train policies robust to model misspecification before fine-tuning them on the higher-fidelity simulation or the real furnace, which is precisely the pipeline that has proven most effective in robotics.

## 7.4 Conclusion

The three contributions described in this chapter reflect the same underlying concern: that reliable experimental science in deep RL requires deliberate investment in tooling, and that this investment is itself a research contribution. **ProbingEnvironments** addresses the absence of principled integration tests in deep RL codebases; the difficulty of adopting it even within this thesis is a truthful illustration of the integration challenges that still face the field. **AJAX** provides a fast, reliable, and modifiable platform for both average-reward and discounted RL experiments: its end-to-end GPU execution model and seed-parallel design made the large-scale experiments of Chapter 8 tractable where the PyTorch-based approach of Chapter 5 was not. **TargetGym** supplies the physics-grounded, continuing-task environments against which

those algorithms are evaluated. Together, they display the software engineering approach to RL research that we wanted to transpire in this thesis, even when the practical constraints of this PhD left some of that vision unrealized.

Beyond the immediate experimental scope of this thesis, the stack assembled here opens the door to the massively parallel training regime discussed in Chapter 6. AJAX's end-to-end GPU execution model and JAX-native architecture mean that scaling from the tens of parallel environments used in this work to the thousands required for sim-to-real transfer is primarily an engineering decision, not a fundamental redesign. The combination of AJAX and TargetGym therefore represents a concrete starting point for future work along that route.

# Part IV

# Reinforcement Learning for Setpoint tracking

# Chapter 8

# Expert-guided Deep Reinforcement Learning

## for Setpoint Tracking

> « Il semble que la perfection soit atteinte, non quand il n'y a plus rien à ajouter, mais quand il n'y a plus rien à retrancher. »
> *"It seems that perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away."*
>
> Terre des hommes — Saint-Exupéry (1939)

Many control tasks in industrial, autonomous systems, and robotic domains involve SPT. Classical approaches such as PID controllers and MPC handle maintenance effectively but their efficiency drops quickly when far from their setpoint, or are limited by the quality of the model they use. RL offers a promising framework to address this setting and the limitations of traditional approaches. The industrial use-case we study in this thesis is a prime example of SPT. However, the SPT setting and its applications remain underexplored in the RL literature, as most algorithms and benchmarks focus on episodic return maximization or terminal-state reaching, rather than stability around a setpoint. Moreover, part of the SPT setting for industrial use cases involves the existence of an existing and successful controller. This means that to provide meaningful contributions to this setting, a solution must not only perform well, but should perform better than the existing solution. In the example of the glass furnace or in the case of the proxy airplane problem, the existing controllers are already performing relatively close to the theoretical maximum performance. Yet, even if existing methods achieve near maximal performance, improving performance by a few points is still significant in those tasks.

In this chapter, we formalize the SPT class of problems as target MDPs, and we build on top of this structure to propose Expert Decayed-Guided-Exploration (EDGE), a method that integrates partial experts into RL agents: We leverage the partial expert knowledge to improve exploration by randomly picking either the expert action or the policy action. This yields policies more asymptotically performant and more sample efficient than their base algorithm. We experimentally validate this method using the Plane task from TargetGym described previously in Section 7.3, while comparing it against similar methods. Finally, to prepare for real-world applications, we measure the sensitivity of this method to imperfections in the expert.

## Contents


**8.1 Introduction** . . . . . **131**
**8.2 Related Work** . . . . . **132**
**8.3 Formalization** . . . . . **134**
**8.4 Method** . . . . . **137**
**8.5 Experiments** . . . . . **142**
**8.6 Conclusion** . . . . . **153**

## 8.1 Introduction

A large fraction of real-world control tasks can be described as problems of reaching and then maintaining the system within a subset of desired states, such as keeping the temperature of a furnace at a target level, regulating the altitude of an aircraft, or balancing a chemical process. This setting is commonly referred to as SPT. Classical control approaches, including PID controllers (Ziegler and Nichols, 1942; Åström and Hägglund, 2001; Ang, Chong, and Y. Li, 2005) and MPC (Richalet et al., 1978; Qin and Badgwell, 2003), have proven remarkably effective in such domains, as they can exploit structural knowledge of the dynamics to achieve stable regulation. However, while these controllers are well-suited for maintaining stability once the system is near the target, they are less effective at guiding the system efficiently the farther the system is from the setpoint (Bloor, Ahmed, et al., 2024; Yuting Wu et al., 2021), or may too heavily rely on a model, and be limited by its accuracy (Forbes et al., 2015). Reinforcement Learning (RL) (Sutton and A. Barto, 2020) provides a natural framework to address these limitations, as it optimizes sequential decision-making through trial and error by leveraging interactions with the environment, avoiding reliance on a flawed model of the environment.

A subtlety specific to industrial SPT is that a working controller is almost always already deployed. PIDs and MPCs are mature technologies, and by the time an organization considers replacing them, they are typically well-tuned and performing near the best achievable for the task. This means the relevant question is not whether RL can learn to control the system at all, but whether it can do so *better than a controller that is already good*. Improvement is therefore measured in small fractions of the remaining headroom between the existing controller and the theoretical optimum, and those fractions translate directly into energy savings, product quality, or safety margins at industrial scale. This framing is fundamentally different from standard RL benchmarks, where the baseline is a random agent and the room for improvement spans orders of magnitude.

Despite this promise, widely studied RL benchmarks and algorithms are not designed toward this particular setting: popular tasks such as MuJoCo locomotion tasks (Todorov, Erez, and Tassa, 2012), Deepmind Control Suite (DMC, Tunyasuvunakool et al. (2020)), or Atari games (Marc G. Bellemare et al., 2013) focus on the cumulative return or reaching a terminal-state. In contrast, many real-world control problems correspond to what we dub *Setpoint Markov Decision Processes* (*SP-MDPs*): problems where the objective is to reach a target region of the state space and maintain the system there as precisely as possible. This class of problems is ubiquitous in applications, yet under-represented in the RL literature. We believe that this under-representation has resulted in a lack of methods tailored to their structure, despite the widespread benefits they could lead to.

To address this issue, in this chapter, we:

- introduce and formalize the concepts of SP-MDPs, and stable policies.
- provide theoretical results regarding the optimality of stable policies in stable-SP-MDPs.
- propose Expert Decayed-Guided-Exploration (EDGE), an RL method that incorporates knowledge of expert stable policies to improve sample efficiency and accelerate convergence toward the target region.
- experimentally validate EDGE against SAC and two other expert-guided baselines, including an ablation study of EDGE mechanisms, and a measure of the robustness of the methods against expert-suboptimality.

## 8.2 Related Work

In this section we will first go through the most relevant pieces of work regarding existing traditional solutions to SPT problems. We will then present expert-based methods by focusing on Imitation Learning (IL) and Residual Reinforcement Learning (Residual-RL). Finally, we will provide an overview of RL applications in industrial processes.

### 8.2.1 Reach-then-maintain

The reach-then-maintain objective underlying the concept of SP-MDP connects to several established strands of research, although no prior work has provided a canonical formalization tailored to reinforcement learning. In the stochastic reachability and reach-avoid literature, the goal is to steer a stochastic dynamical system toward a designated target set while avoiding unsafe regions. Optimal policies and value functions for such problems are typically derived via dynamic programming, partial differential equation methods, or convex optimization approaches such as linear programming (D. P. Bertsekas and Shreve, 1978; Abate et al., 2008; Hernández-Lerma and Lasserre, 2012). In control theory, the complementary "maintain" requirement is addressed through the theory of invariant sets and viability kernels. In particular, control-invariant and robust control-invariant sets characterize the largest subsets of state space from which admissible controllers can keep trajectories inside indefinitely (Blanchini, 1999; Kolmanovsky and Gilbert, 1998). In the field of formal verification and synthesis, temporal logics such as Linear Temporal Logic (LTL) naturally encode reach-and-stay objectives (e.g., the formula "eventually reach $T$, and always remain in $T$ thereafter"). Synthesis techniques typically construct a product MDP between the system and an automaton representing the specification, then optimize satisfaction probability using standard dynamic programming or learning algorithms (Kress-Gazit, Fainekos, and Pappas, 2009; Baier and Katoen, 2008; Fu and Topcu, 2014). By contrast, in reinforcement learning, the study of reach-and-maintain objectives is comparatively limited. Some recent works impose stability constraints when solving MDPs, for example by integrating MPC–style structure or Lyapunov conditions into learned policies

(Berkenkamp et al., 2017; Brunke et al., 2022; Chow et al., 2018). These approaches highlight the importance of stabilization, but they do not formalize a general problem class coupling the tasks of reaching and indefinite maintenance.

### 8.2.2 Imitation Learning

#### Imitation Learning with Suboptimal Experts

Imitation Learning (IL) (Hussein et al., 2017; Osa et al., 2018) trains agents from demonstrations. Traditional IL approaches such as Behavioral Cloning suffer from distributional shift and compounding errors when demonstrations are imperfect. Recent methods explicitly address suboptimality: ILDE introduces dual exploration to achieve beyond-expert performance (Zhao et al., 2025), ROIDA leverages auxiliary data for robustness (Brown et al., 2019), TRAIL learns latent action spaces from imperfect data (Hoang et al., 2024), and ILMAR meta-learns to rank mixed-quality demonstrations (Z. Li, 2022). These works show that surpassing imperfect experts is possible, yet they assume experts provide broad coverage. In contrast, in SP-MDPs, trivial stable policies serve as provably optimal experts on a restricted subset of states, offering a unique structure that differs from classical IL settings.

#### Imitation Learning with Optimal Experts

Another family of IL algorithms relies on optimal experts. These optimal-experts IL methods are mainly targeted at tasks where humans are considered as optimal experts, and the goal is to learn how to reproduce their behavior. Important examples of algorithms belonging to this class are Dataset Aggregation (DAgger, Ross, Gordon, and Bagnell (2011)) and Generative Adversarial Imitation Learning (GAIL, Ho and Ermon (2016)). These methods are, on paper, of little interest to our current application, due to our lack of optimal expert. However, their ways of leveraging expert knowledge can still be useful to us when it is not inherently exploiting the optimality of the expert. For example, the closest algorithm to our proposed method that we could find is IBRL(Hu, Mirchandani, and Sadigh, 2023). In IBRL, the algorithm first learns an expert from optimal demonstrations. This cloned expert is then frozen, and combined with a learned policy by comparing their actions at each timestep and selecting the one with the highest Q-value. Their algorithm is built onto TD3 and stores the executed action into the replay-buffer, enabling learning from both expert and policy actions. In addition, for TD target computations during critic updates, the action for the next-state is also picked as the best action between the current policy and the expert. Due to the close proximity of this algorithm to ours, we will use it as a baseline for comparisons.

### 8.2.3 Residual Reinforcement Learning

Residual-RL (T. Silver et al., 2018; Johannink et al., 2019) combines the strength of existing controllers and the flexibility of RL by using the expert action as a base and learning the residuals to add to this action. This allows starting from an already performant policy, and somewhat benefiting from the safety of its behavior. However, this comes at the cost of a strong bias toward the expert, which may be largely suboptimal in some areas. Residual-RL is a similar concept to the one we study in this chapter; while we do not use a residual policy, it is a strong baseline to consider.

### 8.2.4 RL for Industrial Process Control

RL has seen growing application in process industries for maintaining systems within desired states. Examples include plasma control in nuclear fusion (Degrave et al., 2022), offline RL for flatness control in strip rolling (Deng et al., 2023), chemical process control (Bloor, Ahmed, et al., 2024), and benchmark platforms such as PC-Gym (Bloor, Torraca, et al., 2026) for industrial process control. The works that are closest to our approach are (1) Yuting Wu et al. (2021) that uses PIDs for "steady-state" performance, and RL for the transient performance, and (2) Witman et al. (2019) that uses RL for setpoint tracking. However (1) does not use end-to-end RL, and neither of (1) nor (2) use expert policies. Surveys highlight both promises and challenges in industrial deployment (O. Dogru et al., 2024). While these works emphasize the importance of stability and regulation, neither of them formalize the general setting of SP-MDPs, nor do they propose methods that exploit existing expert policies.

## 8.3 Formalization

Informally, a setpoint tracking task is one where the agent must steer the system toward a designated target region of the state space and keep it there indefinitely. Unlike goal-reaching tasks, where success is declared upon arrival, the difficulty here lies in **maintaining** proximity to the target: the agent receives higher reward the closer the system is to the target, and must continuously act to counteract disturbances or drift. The challenge is therefore not a one-time achievement but a sustained regulation effort, which makes the average-reward framework a natural fit.

We formalize **Setpoint tracking tasks** via **SP-MDPs** in an AR-RL setting, where rewards encourage convergence to a designated set of target states. We then define stability at both the MDP and policy level.

### 8.3.1 SP-MDPs

**Definition 8.1** SP-MDP

An MDP $\mathcal{M}$ is a *SP-MDP* if there exists at least one state coordinate $i_{target}$, a pseudo-distance $d$[a], and a target $x_{i_{target}}$ such that for all $s, s' \in \mathcal{S}$:

$$d(s_{i_{target}}, x_{i_{target}}) > d(s'_{i_{target}}, x_{i_{target}}) \implies r(s) < r(s').$$

[a] Here we do not have $d(x, y) = 0 \implies x = y$, two different states can yield the same reward

For the sake of simplicity in this work, we will only consider the case of a single state coordinate that satisfies this definition (e.g., the target in the plane task described in Section 7.3 is only defined in terms of altitude). This dimension $i_{target}$ is called the **target-dimension**.

Intuitively, the reward increases monotonically as the system approaches the target in at least one dimension. This structure captures practical tasks such as regulating furnace temperature, maintaining vehicle speed, or holding a chemical concentration near equilibrium.

**Remark 8.1.**

Definition 8.1 implicitly requires $\mathcal{R}$ to be upper-bounded, i.e., $r^* = \max(\mathcal{R})$ must be finite. This is a mild assumption satisfied by any finite-valued reward signal in practice. Moreover, in the applications that SP-MDPs are designed to model, the reward function is typically user-defined (e.g., a distance to a setpoint), so $r^*$ is not merely finite but explicitly known. This makes SP-MDPs unusual among RL settings in that the theoretical performance ceiling is available by construction.

**Definition 8.2** Target States

In a SP-MDP, the *target states* are defined as

$$\mathcal{S}_{target} = \{s \in \mathcal{S} \mid r(s) = r^*\}, \text{ with } r^* = \max(\mathcal{R}).$$

and their $\varepsilon$-relaxation is

$$\mathcal{S}_{\varepsilon\text{-target}} = \{s \in \mathcal{S} \mid r^* - r(s) \leq \varepsilon\},$$

allowing for approximate stabilization within a tolerance $\varepsilon$.

Unlike many RL settings where the theoretical performance ceiling is unknown, this upper bound is given directly by the MDP structure. While it is not exploited algorithmically by

our method, it enables the construction of optimality-gap metrics that express an agent's performance relative to both the theoretical ceiling and a reference expert policy, as we formalize in Section 8.5 through the Normalized Expert Advantage (NEA).

**Definition 8.3** Stable-SP-MDP

A SP-MDP is a *Stable-SP-MDP* (ST-MDP) if there exists a policy $\pi \in \Pi$, a timestep $T$, and a subset $\mathcal{S}' \subseteq \mathcal{S}_{target}$ such that $\forall s \in \mathcal{S}', \forall t > T$:

$$P(s_t, \pi(s_t)) \in \mathcal{S}'.$$

Stable-SP-MDPs, by definition, represent tasks where the existence of a policy that can reach and maintain the target states is guaranteed. In the applications we study, this is a realistic hypothesis (up to some relaxation on the distance to the target states), as the current solutions (PID/MPC) already achieve this goal to a degree.

**Definition 8.4** Stable Policy

A policy $\pi$ is *stable* in a SP-MDP, if

$$\lim_{t\to\infty} d(s_{t+1,i_{target}}, s_{t,i_{target}}) = 0, \text{ with } s_{t+1} = P(s_t, \pi(s_t))$$

.

A stable policy reaches and maintains a certain level in the target-dimension of the SP-MDP. The set of states that share this value is called $\mathcal{S}_\pi$.

Stable policies thus guarantee convergence and indefinite maintenance within a certain region of the state-space, aligning with Lyapunov-style stability (Lyapunov, 1992) in control theory.

### 8.3.2 Optimality and Stability

Because SP-MDPs are formalized as continuing tasks, the natural optimality criterion is the average-reward one. We briefly recall the relevant notions here.

**Definition 8.5** Gain and Bias

The *gain* of a policy $\pi$ from state $s$ is the long-run average reward:

$$\rho^\pi(s) \triangleq \lim_{N\to\infty} \mathbb{E}\left[\frac{\sum_{t=0}^{N-1} r_t^\pi}{N} \,\middle|\, s_0 = s\right].$$

The *bias* of $\pi$ from $s$ captures the transient cumulative deviation from the gain:

$$b^{\pi}(s) \triangleq \lim_{N\to\infty} \mathbb{E}\left[\sum_{t=0}^{N-1} \left(r_t^{\pi} - \rho^{\pi}(s)\right) \,\middle|\, s_0 = s\right].$$

**Definition 8.6** Gain-Optimality and Bias-Optimality

A policy $\pi^*$ is *gain-optimal* iff $\rho^{\pi^*}(s) \geq \rho^{\pi}(s), \forall \pi, \forall s \in \mathcal{S}$.
A policy $\pi^*$ is *bias-optimal* iff it is gain-optimal and $b^{\pi^*}(s) \leq b^{\pi}(s), \forall \pi, \forall s \in \mathcal{S}$.

Gain-optimality captures long-run performance; bias-optimality additionally minimizes transient deviations from the gain, selecting among equally good long-run policies the ones that converge fastest to the high-reward region. A natural question is whether optimal policies in SP-MDPs must also be stable.

**Theorem 8.1** Gain-Optimality in Stable-SP-MDPs.

*If $\mathcal{M}$ is a Stable-SP-MDP, all stable-policies $\pi$ such that $\mathcal{S}_{\pi} \subseteq \mathcal{S}_{target}$ are gain-optimal.*

**Proof.** See Appendix C.1.1 □

**Theorem 8.2** Bias-Optimality in Stable-Target MDPs.

*If $\mathcal{M}$ is a Stable-SP-MDP, then all bias-optimal policies in $\mathcal{M}$ are stable policies.*

**Proof.** See Appendix C.1.2. □

This result ensures that we can restrict ourselves to consider only stable-policies during learning, without missing any bias-optimal candidate. Furthermore, in a Stable-SP-MDP (definition 8.3), a stable expert already exists by definition: a policy that reaches and maintains the system within target states, operating precisely in the high-reward region of state space. This makes it a principled source of exploration guidance for any learning agent operating in the same MDP.

## 8.4 Method

From theorem 8.2 we know that, in a SP-MDP, all bias-optimal policies are stable policies. This observation motivates us to restrict policy search to stable behaviors. However, designing

neural parameterizations that guarantee stability requires specialized architectures (e.g., neural Lyapunov methods (Chang, Roohi, and Gao, 2019)) that do not scale to general MDPs, as stability is a global property of closed-loop dynamics rather than a local constraint on the policy class. Crucially, in a Stable-SP-MDP, a stable expert already exists by definition: one that reaches and maintains the system within target states and thus operates in the high-value region where bias-optimal policies are concentrated. Rather than constraining the policy class directly, we propose to leverage this expert as a source of exploration guidance: by occasionally executing its actions, we steer the agent toward the region of the state space where an optimal behavior is known to lie. While residual-RL leverages the expert by learning on top of its action, and IL learns to imitate the expert policy, we propose to go a different route, purely based on exploration. Figure 8.1 illustrates the core intuition.

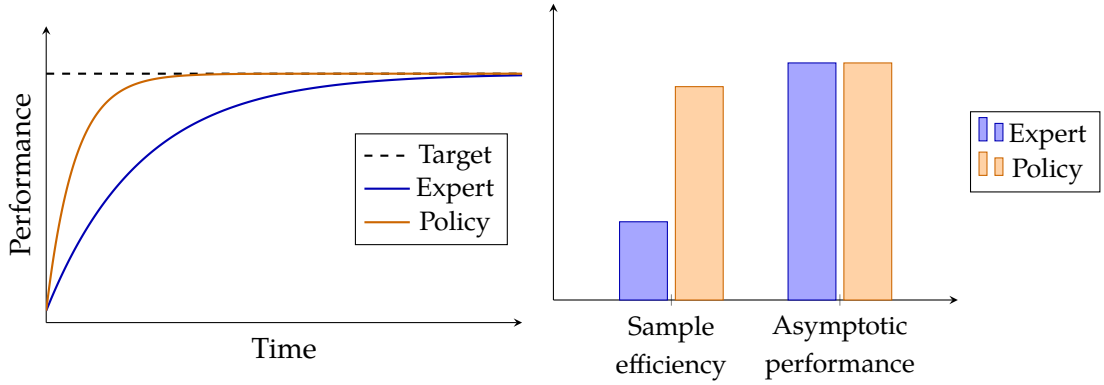



**Figure 8.1** – *Conceptual illustration.* The expert (blue) represents a stable policy that converges toward the target (dashed) but does so slowly; the learned policy (orange) reaches the same target faster, and both ultimately stabilize at the same level. *Left*: convergence over time. *Right*: the difference in sample efficiency is large (the policy reaches the target much sooner), while the asymptotic performance is the same for both. The expert's value lies in guiding early exploration toward the high-reward region; the policy is then free to surpass it in convergence speed.

### 8.4.1 Expert Decayed-Guided-Exploration (EDGE)

EDGE is our proposed method for leveraging a stable expert in a SP-MDP to improve exploration in off-policy RL. The core idea is simple. At each environment step, instead of always executing the policy's action, we flip a biased coin: with probability $p_t$ we execute the expert's action, and with probability $1 - p_t$ we execute the policy's action. The resulting transition, whichever action was picked, with the reward and next state it actually produced, is stored in the replay buffer and used for standard updates. EDGE is therefore designed for **off-policy** algorithms; the data collection modification operates at the level of the replay buffer, and the design choices discussed in Section 8.4.2 rely on this structure. Extending it to on-policy algorithms would require separate treatment.

Formally, at step $t$ of a training run of total length $T$, the gate probability is:

$$p_t = \varepsilon \cdot \max\left(1 - \frac{t}{T_{\text{dec}} \cdot T},\ 0\right),$$

where $\varepsilon \in (0, 1)$ is the mixing probability and $T_{\text{dec}} \in [0, 1]$ is the decay fraction controlling how long the expert remains active. The expert is sampled with probability $p_t$, the policy with probability $1 - p_t$. After $T_{\text{dec}} \cdot T$ steps, $p_t = 0$ and the policy acts alone for the remainder of training.

Since EDGE is designed as a wrapper around any off-policy algorithm, the only modification to the base algorithm is in the data collection step. The replay buffer receives transitions of the form $(s_t, a_t, r_t, s_{t+1})$ where $a_t$ is the action that was actually executed, either $\pi^e(s_t)$ or a sample from $\pi_\theta(\cdot \mid s_t)$, depending on the coin flip. All downstream updates are standard and require no modification. As an example, we will apply the EDGE modifications to SAC. The full algorithm is given in algorithm 5.

It is worth noting that, while the theoretical motivation assumes the expert is a stable policy in a Stable-SP-MDP, the algorithm itself makes no such verification. The expert is treated as a black-box policy: any policy can serve as the expert, and EDGE will still run. A stable expert is expected to work best, since it concentrates experience in the high-reward region of state space as the theory suggests, but the degradation study in Section 8.5.4 shows that the method remains robust even when the expert is substantially corrupted. This generality is a practical advantage, at the cost of weakening the theoretical justification: the framing of SP-MDPs motivates the approach, but does not strictly constrain it.

**Algorithm 5.** Expert Decayed-Guided-Exploration SAC (EDGE-SAC)

**Initialize** policy $\pi_\theta$, critics $Q_{\phi_1}$, $Q_{\phi_2}$, targets $\bar{Q}_{\phi_1} \leftarrow Q_{\phi_1}$, $\bar{Q}_{\phi_2} \leftarrow Q_{\phi_2}$, buffer $\mathcal{D}$, temperature $\alpha$
**Require** expert policy $\pi^e$, $\varepsilon \in (0,1)$, $T_{\text{dec}} \in [0,1]$
**for** $t = 1, \dots, T$ **do**
  *Action selection*
  $p_t \leftarrow \varepsilon \cdot \max\big(1 - t \,/\, (T_{\text{dec}} \cdot T),\ 0\big)$ ▷ expert probability decay
  **if** $\text{Bernoulli}(p_t) = 1$ **then** $a_t \leftarrow \pi^e(s_t)$ **else** $a_t \sim \pi_\theta(\cdot \mid s_t)$
  Execute $a_t$, observe $r_{t+1}$, $s_{t+1}$
  Store $(s_t,\ a_t,\ r_{t+1},\ s_{t+1})$ in $\mathcal{D}$ ▷ executed action stored (expert or policy)
  **– Critic update –**
  Sample batch $\{(s, a, r, s')\} \sim \mathcal{D}$
  $a' \sim \pi_\theta(\cdot \mid s')$
  $y = r + \gamma\Big[\min_{i=1,2} \bar{Q}_{\phi_i}(s', a') - \alpha \log \pi_\theta(a' \mid s')\Big]$
  $\phi_i \leftarrow \phi_i - \nabla_{\phi_i}\big(Q_{\phi_i}(s,a) - y\big)^2$ for $i = 1, 2$
  **– Actor update –**
  $\tilde{a} \sim \pi_\theta(\cdot \mid s)$
  $\theta \leftarrow \theta + \nabla_\theta[\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}) - \alpha \log \pi_\theta(\tilde{a} \mid s)]$
  **– Temperature update –**
  $\alpha \leftarrow \alpha - \nabla_\alpha\Big[\alpha\Big(-\log \pi_\theta(\tilde{a} \mid s) - \bar{\mathcal{H}}\Big)\Big]$
  **– Target networks update –**
  $\bar{Q}_{\phi_i} \leftarrow \tau\, \bar{Q}_{\phi_i} + (1 - \tau)\, Q_{\phi_i}$ for $i = 1, 2$
**end for**

### 8.4.2 Design Choices and Intuitions

The method as stated involves four choices, each of which could have been made differently. We explain each of them.

**Stochastic mixing vs. argmax selection.** The natural alternative is to select the action with the higher Q-value at each step, which is the approach used by IBRL (Hu, Mirchandani, and Sadigh, 2023). This seems appealing: defer to the expert only when the critic says it is genuinely better. However, this value-gated selection causes the kind of extrapolation error identified in offline RL (Fujimoto, Meger, and Precup, 2019), where the critic is queried on actions it has rarely or never seen during training.

The mechanism is as follows. Assume, as is the premise for using an expert at all, that the expert outperforms the initial policy. Then early in training, the set of states where the expert's

Q-value exceeds the policy's,

$$\mathcal{M}_t = \left\{ s : \min_i Q_{\phi_i}(s, \pi^e(s)) > \min_i Q_{\phi_i}(s, \pi_\theta(s)) \right\},$$

covers most of the state space. At states in $\mathcal{M}_t$, argmax always selects the expert, so the replay buffer accumulates few policy-action transitions there. When the actor update later queries $Q_{\phi_i}(s, \tilde{a})$ for $\tilde{a} \sim \pi_\theta(\cdot \mid s)$, the critic must evaluate a policy action at a state where it has only been trained on expert-action transitions: an out-of-distribution query that introduces a systematic bias in the actor gradient. As training progresses and the policy improves, $\mathcal{M}_t$ shrinks, but the early bias has already shaped the learned policy. In SP-MDPs, where the expert is already near the performance ceiling, $\mathcal{M}_t$ is large precisely during the phase when the actor gradient matters most.

Stochastic mixing sidesteps this by ensuring that at every state, the policy has a nonzero probability of acting, regardless of the current value gap. The buffer therefore contains policy-action transitions across the full state distribution, keeping critic queries in-distribution and the actor gradient unbiased. We verify empirically that this matters in the ablation study (Q3, Section 8.5).

**Storing the executed action** When the expert acts, we store the expert's action in the replay buffer alongside the resulting reward and next state. An alternative would be to store the policy's action instead, effectively treating the expert's intervention as invisible to the learning algorithm. This is how DAgger handles the analogous situation in imitation learning. We deliberately do not do this.

Storing the executed action means the critic observes that expert transitions have high reward and high subsequent value. This creates a direct learning signal: the critic learns accurate Q-value estimates in the high-value regions of state space that the expert populates, and these estimates propagate to the policy via the actor update. If we stored the policy's action instead, the critic would never observe what happens when those expert actions are actually taken, and the buffer enrichment effect would be lost.

**Global decay** The expert acts with a probability that decays to zero at step $T_{\text{dec}} \cdot T$, after which the policy has full autonomy. We found through ablations that the decay horizon is the single most important hyperparameter of the method, and that the relationship is non-trivial: too short a decay (removing the expert before the critic has calibrated) is nearly as bad as no expert at all, while too long a decay (or no decay) degrades asymptotic performance by permanently contaminating the replay buffer with transitions from a policy that becomes suboptimal relative to the learned agent.

The intuition for why permanent expert mixing hurts is the following: Early in training, the expert transitions are more structured and higher-value than the policy near-random transitions, which helps the critic calibrate faster. But once the policy has surpassed the expert, continuing to inject expert transitions introduces data from an inferior policy. The replay buffer now contains a mixture of the agent's own high-quality experience and the expert's comparatively lower-quality reach-phase trajectories, which pulls the critic value estimates downward in the states where the agent has learned to outperform the expert. The global decay removes this contamination at the right time.

**Fixed mixing probability.** One might expect that using the Q-value gap between the expert policy $\pi^e$ and the learned policy $\pi_\theta$, $Q(s, \pi^e(s)) - Q(s, \pi_\theta(s))$, to modulate the mixing probability would be strictly better than a fixed coin flip. If the critic says the expert is much better, mix more; if it says the policy is better, mix less. From previous experiments, we have so far not managed to produce better results with this value-dependent gating than with the fixed mixing probability, and have resolved to keep the simpler fixed approach for now. We believe the reason of Q-value dependent gating underperforming is timing. The value gap signal is most informative late in training, when the critic is well-calibrated and the gap reliably reflects genuine performance differences. But the expert mixing is most impactful early in training, precisely when the critic is miscalibrated and the gap is noisy. In the window where the gate matters most, it is operating on unreliable estimates. A fixed probability is simply more stable over that critical early window. We note that a principled fix exists: using critic ensemble disagreement to modulate the temperature of the value-gap gate, so the gate behaves like a coin flip when the critic is uncertain and sharpens toward argmax as the critic calibrates.

## 8.5 Experiments

### 8.5.1 Experimental setup

#### Environment and expert

To evaluate the impact of EDGE on SAC we chose to use our plane environment from TargetGym described in Section 7.3. This choice was first and foremost motivated by its proximity with our industrial use-case; but also for the ease of identification of trivial partial experts (that are empirically proven to maintain the plane in target states) in the form of constant actions. The process to identify these trivial stable policies is described in Appendix C.2 as it is not part of the EDGE method itself. In our method, we assume the expert is provided by the user. The task hyperparameters are kept to the default values, as presented in Section 7.3.

### Baselines and hyperparameters

Once the task is identified, the second task is to select relevant baselines:

- For the expertless-baseline, as we are dealing with continuous actions, either SAC or TD3 are good candidates. We chose to use SAC for its larger adoption as a baseline in the literature.
- As discussed in Section 8.2.2, we will also compare to IBRL as we use the same core idea of expert and policy mixing, with different mechanisms. As IBRL uses TD3 as a base, we implement a SAC version instead. To isolate the comparison to the action selection mechanism, we restrict our IBRL baseline to this modification only, omitting the TD-target bootstrap described in Section 8.2.2 (where the next-state action in critic updates is also chosen as the best of expert and policy). Including this additional critic modification would conflate two sources of difference, making it impossible to attribute performance gaps specifically to the gating mechanism. We empirically verify in Appendix C.4 that this omission is justified: comparing all four combinations (IBRL and EDGE each with and without bootstrap) shows no measurable difference across any metric.
- As discussed in Section 8.2.3, we add residual-RL as our last baseline. This is not due to direct method similarity, but rather a similarity in design philosophy: leveraging expert knowledge. However, residual-RL differs significantly in the way the expert is leveraged, as it is intrinsically part of the policy.

For the pseudocode of the EDGE version of the baselines we used, please refer to Section C.3. For all experiments in this chapter we use the AJAX library, as described in Section 7.2.

Once the baselines have been selected, we run a hyperparameter search for SAC on our plane task, and use the resulting hyperparameters for all algorithms. Our reasoning is that all the methodologies we compare against SAC are extensions of the base algorithm, and should therefore be compared as such, with modifications limited to the extension hyperparameters as degrees of freedom. Better results could naturally be achieved if the base SAC hyperparameters were tuned specifically for each of the other methods; however, that would require significantly more computational budget for hyperparameter search, without necessarily altering the conclusions of this chapter. For the hyperparameter search methodology, please refer to Appendix C.3, and for the hyperparameters used for all methods refer to Appendix C.3.

### Metrics and statistics

Finally, we need to pick the relevant metrics to collect, and the most adequate summary statistics to represent them. As the goal of our method is to try and improve both sample efficiency and asymptotical performance, we chose:

**Performance** For performance, we report the Normalized Expert Advantage:

**Definition 8.7** Normalized Expert Advantage

The NEA at timestep $t$ is the difference between the agent's mean episodic cumulative return and the expert's mean episodic cumulative return, normalized by the difference between the maximum theoretical return and the expert's mean episodic cumulative return:

$$\text{NEA}(t) = \frac{\mathbb{E}_{\tau\sim\pi_\theta}\left[G(\tau)\right] - \mathbb{E}_{\tau\sim\pi^e}\left[G(\tau)\right]}{G_{\max} - \mathbb{E}_{\tau\sim\pi^e}\left[G(\tau)\right]}$$

where $\pi_\theta$ is the agent policy, $\pi^e$ is the expert policy, $G(\tau) = \sum_{k=0}^{T} r_k$ is the episodic cumulative return of trajectory $\tau$, and $G_{\max}$ is the maximum theoretical episodic return.

We compute the Interquartile Mean (IQM) of NEA over the last 200,000 timesteps as our metric of choice.

**Sample efficiency** For sample efficiency we report the number of timesteps to reach various levels of NEA: $0\%$, $20\%$, and $40\%$. All metrics are collected every 5000 steps and aggregated over 25 evaluation episodes, with deterministic action selection (no exploration). All algorithms are trained for $10^6$ timesteps.

For the choice of summary statistics, we follow the statistical methodology proposed by Agarwal et al. (2021), using the IQM combined to bootstrap Confidence Interval (CI) as our statistics of choice. In addition, for the most important results, we add statistical tests (IQM permutation test + Holm-Bonferroni) to compare EDGE to each of the other baselines, please refer to Appendix C.5 for the methodology.

All results reported for this work were aggregated over 100 random seeds.

### 8.5.2 Main Results

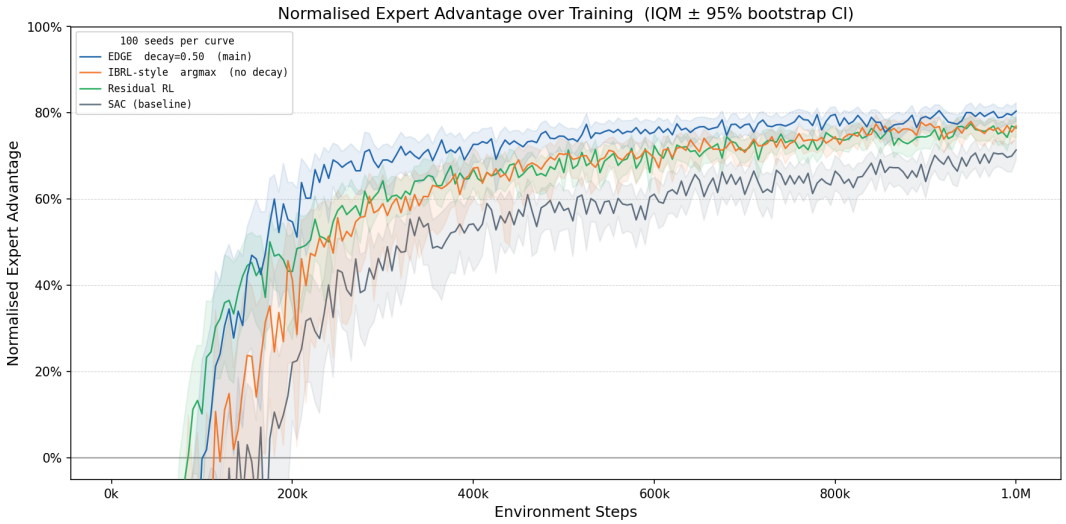



**Figure 8.2** – Normalized Expert Advantage (NEA) over training, aggregated as IQM across 100 seeds. Shaded bands are 95% bootstrap confidence intervals (1 000 resamples). Higher is better; NEA = 0 indicates the policy matches the expert.

In Figure 8.2 we can observe that all methods follow the same pattern of quick rise and plateau, clearly showing the saturated nature of the task. As expected from the saturation effect, apart from the SAC baseline which is clearly underperforming, the other methods achieve comparable results. Looking at Figure 8.3 we can more clearly compare the final performance of IBRL, residual-RL and EDGE, and observe that EDGE achieves better asymptotical performance than all other methods.

For the other metrics, as the CI are large relative to the differences between methods, we resorted to permutation tests on the IQM with Holm–Bonferroni correction for multiple comparisons (see Appendix C.5 for the details). The conclusions are:

- For the time to reach the expert performance (NEA=0), EDGE is not significantly faster than any of the baselines ($p_{\min} < 0.1068$).
- For NEA=20%, EDGE is significantly faster than SAC ($p < 0.0039$).
- For NEA=40%, EDGE is significantly faster than SAC and residual-RL ($p < 0.0498$), but not IBRL ($p = 0.2823$).

We kindly remind the reader that not being significantly faster does not imply that one is slower or even as fast, just that there was not enough samples to confidently tell them apart. Last but not least, the test for the final performance confirms what Figure 8.3 showed, EDGE is asymptotically better than all baselines ($p < 0.0091$).

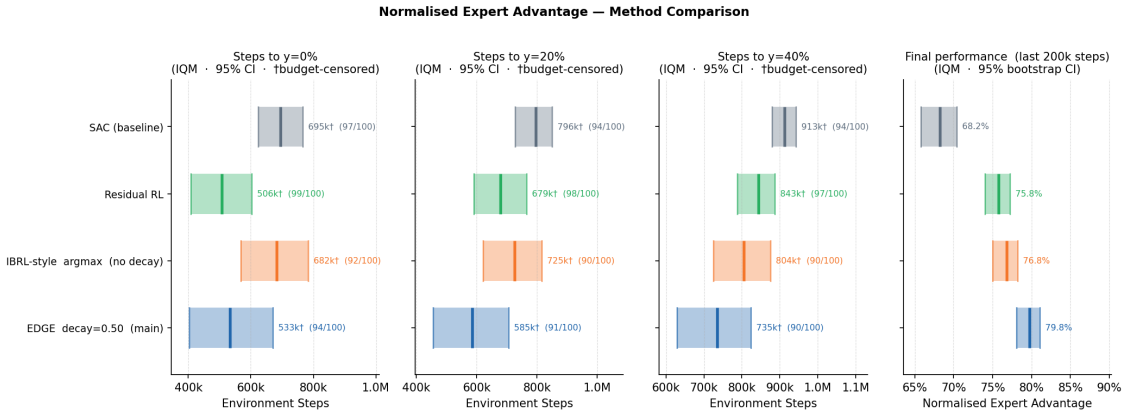



**Figure 8.3** – Summary statistics for the four methods. Each bar shows the IQM (vertical line) and 95% bootstrap CI (shaded rectangle) across 100 seeds. The first three panels report steps to reach NEA $\geq 0\%$, 20%, and 40%; the rightmost panel reports final NEA (IQM over the last 200 000 steps). Seeds that never reach a threshold within the $10^6$-step budget are budget-censored and marked with †.

Table 8.1 summarizes the exact IQM values and relative improvements over SAC. We refer the reader to Figure 8.3 for the confidence intervals, which are essential for interpreting the magnitude of these differences.

| **Method** | Steps to $y = 0$ | | Steps to $y = 20\%$ | | Steps to $y = 40\%$ | | **IQM last 200k steps** | |
|---|---|---|---|---|---|---|---|---|
| | IQM | Δ% | IQM | Δ% | IQM | Δ% | IQM | Δ% |
| SAC (baseline) | 695k† (97/100) | — | 797k† (94/100) | — | 913k† (94/100) | — | 68.2% | — |
| Residual RL | **507k† (99/100)** | -27.1% | 680k† (98/100) | -14.7% | 843k† (97/100) | -7.7% | 75.8% | +11.1% |
| IBRL-style argmax (no decay) | 682k† (92/100) | -1.9% | 725k† (90/100) | -9.0% | 805k† (90/100) | -11.9% | 76.8% | +12.7% |
| **EDGE**(ours) | 534k† (94/100) | -23.2% | **586k† (91/100)** | -26.5% | **736k† (90/100)** | -19.4% | **79.8%** | +17.0% |

† Some runs did not reach this threshold; censored at training budget for IQM. $(X/Y)$ denotes the number of seeds that reached the threshold out of $Y$ total seeds.

**Table 8.1** – Normalized Expert Advantage (NEA) summary. The last column reports the IQM (interquartile mean) of NEA over the last 200k environment steps, computed per seed then aggregated with IQM across seeds.

### 8.5.3 Ablation study

Now that we have validated the performance of EDGE against our baselines, we want to independently measure how each design choice in EDGE contributes to its performance. This translates into answering the following questions:

1. **Q1 – Decay horizon**: How important is the decay on the expert contribution to action selection? We sweep $f_{\text{decay}} \in \{\text{no decay}, 0.05, 0.15, 0.50, 0.75\}$.

2. **Q2 – $\varepsilon$ sensitivity**: How important is the initial probability to select the expert action? We sweep $\varepsilon_0 \in \{0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99\}$, with $f_{\text{decay}} = 0.50$ fixed.
3. **Q3 – Gating style**: What is the best gating mechanism to select between the expert and the policy? We compare stochastic $\varepsilon$-greedy (EDGE), argmax without decay (IBRL-style), and argmax with decay ($f_{\text{decay}} = 0.50$).
4. **Q4 – Expert necessity**: Does EDGE benefit from the expert's domain knowledge, or would decayed random exploration alone yield similar gains? We compare EDGE with the actual stable expert against a variant where the expert is replaced by a uniform random-action policy and the SAC baseline, using fixed hyperparameters ($f_{\text{decay}} = 0.50$, $\varepsilon_0 = 0.50$).

For Q1–Q3, we run sweep experiments with the same hyperparameters as the main results, varying only the hyperparameter under study. Q4 is a single comparison between EDGE with the actual stable expert, EDGE with a uniform random-action policy, and the SAC baseline, using fixed hyperparameters ($f_{\text{decay}} = 0.50$, $\varepsilon_0 = 0.50$). An additional ablation verifying that the omission of the TD-target bootstrap from both IBRL and EDGE does not affect performance is reported in Appendix C.4.

### Q1: Decay horizon

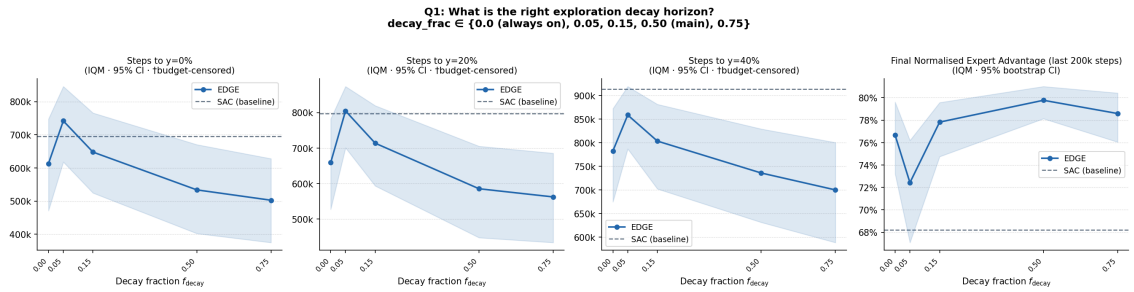



**Figure 8.4** – Q1: Effect of the decay fraction $f_{\text{decay}}$ on EDGE performance (sweep values as in Q1 above), compared against the SAC baseline.

Regarding the importance of decay in EDGE, in Figure 8.4 we can observe that having no decay (i.e., $p_t = \varepsilon$ throughout training) underperforms most decay values. A very short decay of 5% of training time similarly underperforms; both extremes thus behave poorly. A short decay implies that the expert guidance is rendered insignificant before the critic learns accurate Q-values, leaving the agent without expert guidance for most of training; additionally, the few expert actions collected early in the buffer come from a very different distribution than the agent's evolving policy, and appear to hinder learning rather than help. We observe that a longer decay correlates with better sample efficiency (observed through the time to reach the different thresholds). However, there seems to be a threshold (between 50% and 75%) above which longer decay seems harmful to final performance. This is most likely due to

the asymptotically suboptimal actions of the expert still being randomly taken, hindering the trained policy.

**Outtake** Keeping the expert active for a large portion of training is the best choice. This means picking a large decay value, with the added complexity of picking a value small enough not to perturbate the asymptotic policy.

### Q2: $\varepsilon$ sensitivity

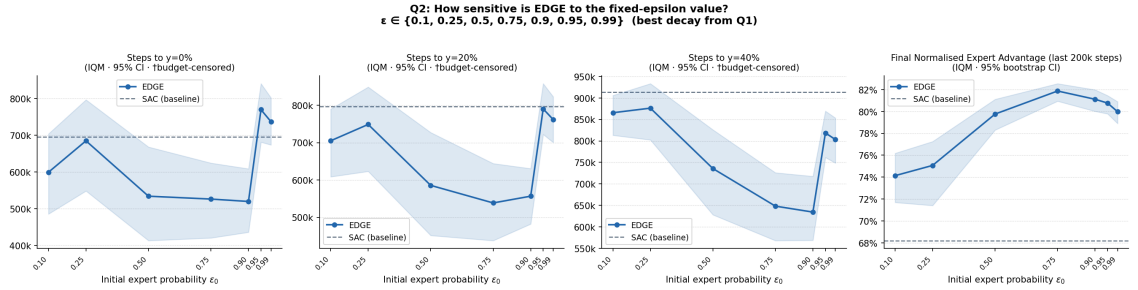



**Figure 8.5** – Q2: Sensitivity of EDGE to the initial expert probability $\varepsilon_0$ (sweep values as in Q2 above). Layout as in Figure 8.4.

Concerning the sensitivity of EDGE to the initial expert probability $\varepsilon_0$, in Figure 8.5 we can observe two failure modes at the extremes. For very low values ($\varepsilon_0 \leq 0.25$), the expert is rarely consulted, yielding little benefit over SAC in both sample efficiency and asymptotic performance; the expert guidance is simply too sparse to meaningfully influence exploration. For very high values ($\varepsilon_0 \geq 0.95$), sample efficiency degrades sharply, with steps-to-threshold returning close to the SAC baseline; this is most likely because the agent takes so many expert actions that it has insufficient opportunity to explore its own policy, slowing down learning. Interestingly, the asymptote remains high even at $\varepsilon_0 = 0.99$, as the expert's stable actions still anchor the policy near good states in the long run. In the middle range $\varepsilon_0 \in [0.50, 0.90]$, EDGE performs consistently well across all metrics, and the overlapping confidence intervals suggest the method is robust to the exact choice of $\varepsilon_0$ within this range.

**Outtake** EDGE is not very sensitive to $\varepsilon_0$ as long as it is chosen in a reasonable middle range. Values that are too low waste the expert, while values that are too high over-constrain early exploration. A value around $\varepsilon_0 = 0.75$–$0.90$ appears to offer the best trade-off between sample efficiency and asymptotic performance.

### Q3: Gating style

In this section we study the importance of the action selection method. As it is naturally coupled to the duration and magnitude of its impact through the decay value, and to provide a fair

comparison, we augment IBRL with decay for an additional baseline. Due to the use of argmax in IBRL, we cannot design a progressive cutoff. Instead the expert guidance is completely shut off at $f_{\text{decay}} = 0.50$ (picked as a good value from the previous question), reverting to normal SAC.

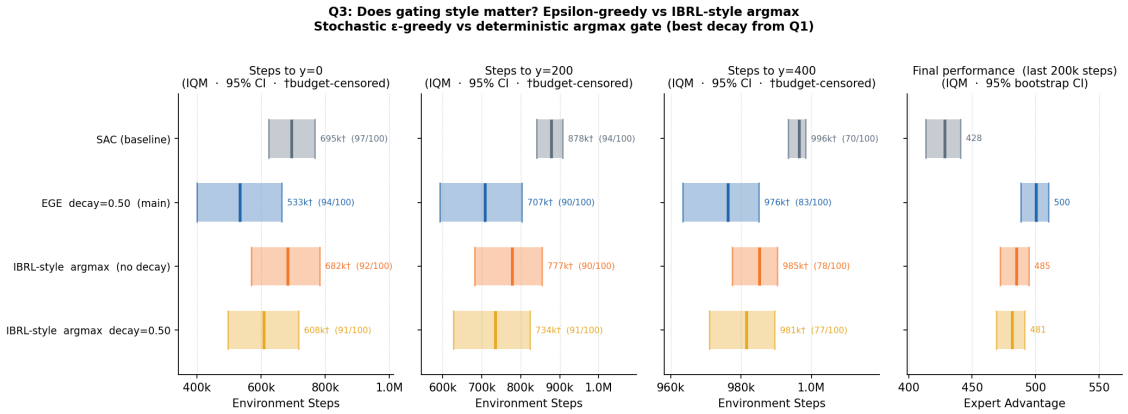



**Figure 8.6** – Q3: Comparison of gating mechanisms (variants as in Q3 above). Layout as in Figure 8.3.

Finally, for the importance of gating style in EDGE, in Figure 8.6 we can observe that the stochastic $\varepsilon$-greedy gating of EDGE consistently outperforms both deterministic argmax variants across all metrics, achieving better sample efficiency at every threshold as well as a higher asymptotic performance ($79.8\%$ vs $76.8\%$ and $76.7\%$). This is most likely because stochastic gating ensures a predictable rate of expert guidance throughout training, regardless of Q-value calibration; in contrast, argmax gating only defers to the expert when $Q(\text{expert}) > Q(\text{policy})$, a condition that becomes increasingly rare as the agent improves and Q-values converge, effectively switching off expert guidance too early. Notably, adding decay to the argmax variant improves its sample efficiency over the no-decay version, which is consistent with the findings of Q1. Meanwhile the asymptotic performance of both argmax variants remains nearly identical ($76.8\%$ vs $76.7\%$), confirming that the gating mechanism itself, rather than the decay schedule, is the primary driver of the asymptotic gap with EDGE.

**Outtake** Gating technique does matter. The stochastic $\varepsilon$-greedy mechanism of EDGE provides more robust and consistent expert guidance than deterministic argmax gating, translating into both faster learning and better final performance. Moreover, early cutoff of expert guidance also benefits IBRL.

### Q4: Expert necessity

A natural alternative explanation for EDGE's gains is that improvement comes not from the expert's domain knowledge itself but simply from the additional exploration the decay schedule provides: any action source, including a random one, might improve over SAC if injected with

the same schedule. To test this, we replace the stable expert with a uniform random-action policy while keeping all other hyperparameters identical. If the schedule alone drives performance, the random variant should match EDGE; if the expert's domain knowledge is essential, the random variant should perform at or below SAC.

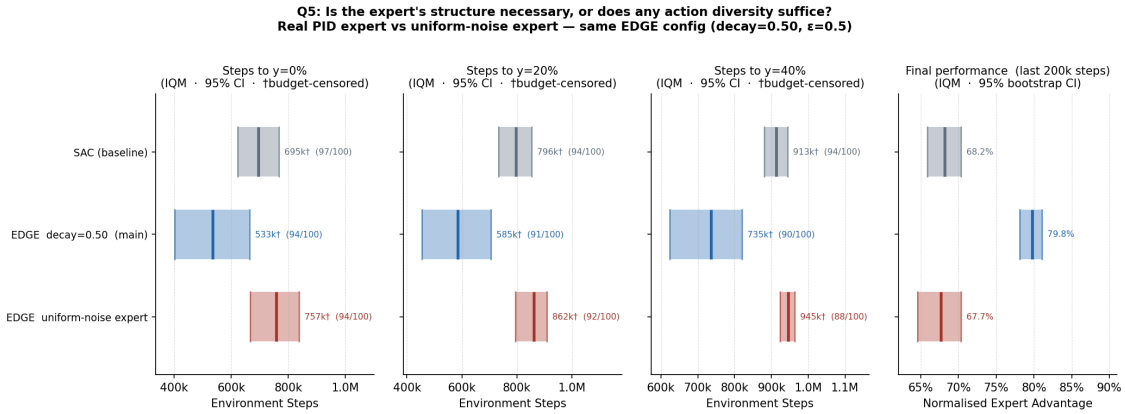



**Figure 8.7** – Q4: Effect of replacing the stable expert with a uniform random-action policy. Layout as in Figure 8.3. The random-expert variant performs at or below the SAC baseline across all metrics, confirming that EDGE's gains stem from the expert's domain knowledge rather than from the exploration schedule alone.

In Figure 8.7, replacing the expert with random actions yields performance at or below the SAC baseline across all metrics. This rules out the alternative hypothesis: SAC's entropy term already provides principled stochastic exploration, so injecting additional random actions from a dummy source only adds noise to the replay buffer without directing the agent toward target states. The improvement observed for EDGE thus stems from the expert's ability to guide early exploration toward the high-reward region of state space, as the theoretical motivation in Section 8.4.1 predicts.

**Outtake** The expert's domain knowledge is necessary for EDGE to work. A decayed random exploration schedule yields no improvement over SAC, confirming that the method's gains arise from the expert guiding the agent toward target states rather than from the exploration schedule itself.

### 8.5.4 Expert suboptimality sensitivity

A key aspect of real-life usefulness of expert-based methods is their robustness to expert suboptimality. Indeed, in real scenarios, the expert might be partially wrong or noisy, and thus methods should still perform under these circumstances to be considered useful outside of perfect experimental research setups. To assess robustness to expert quality, we introduce noise by corrupting the expert perception of the target altitude while leaving its control law intact.

Concretely, the expert observes a noisy target $\tilde{z}^* = \text{clip}(z^* + \nu,\, alt_{\min}, alt_{\max})$, $\nu \sim \mathcal{N}(0, \sigma^2)$, instead of the true target $z^*$, where $\sigma = \rho \times 5000\,\text{m}$ and $\rho$ is the noise level. The clipping is here to prevent requesting targets the expert cannot reach, leading to undefined behavior (NaNs in practice). The goal here is not to simulate noisy observations itself, but rather to have the expert provide actions that are not aiming for the correct target, simulating a suboptimal expert (i.e., a badly tuned PID). While this degradation methodology does not encompass all possible differences between simulation and actual real-life scenarios, we believe it can still shed light on the robustness of algorithms to those Simulation-to-Real (sim2real) discrepancies. We emphasize that **this study was not part of either the IBRL or residual-RL papers, and is thus a novel contribution of this thesis**.

The SAC agent always receives the true target altitude. Seven noise levels are tested, with the following noise ratios and corresponding standard deviations on the perceived target altitude:

$$\rho \in \{0\%,\ 2\%,\ 5\%,\ 10\%,\ 20\%,\ 40\%,\ 80\%\},$$
$$\sigma \in \{0,\ 100,\ 250,\ 500,\ 1000,\ 2000,\ 4000\}\,\text{m}.$$

At $\rho = 0\%$ the expert is noise-free and identical to the one used in the main experiments. At $\rho = 80\%$, the standard deviation of the noise ($4000\,\text{m}$) is comparable to the full altitude range of the task, meaning the expert's perceived target is essentially random and its actions provide no useful signal about the true setpoint.

Each of the three algorithms (EDGE, IBRL,residual-RL) is trained independently at every noise level using the same hyperparameters, budget ($10^6$ steps) and number of seeds ($100$) as the main comparison.

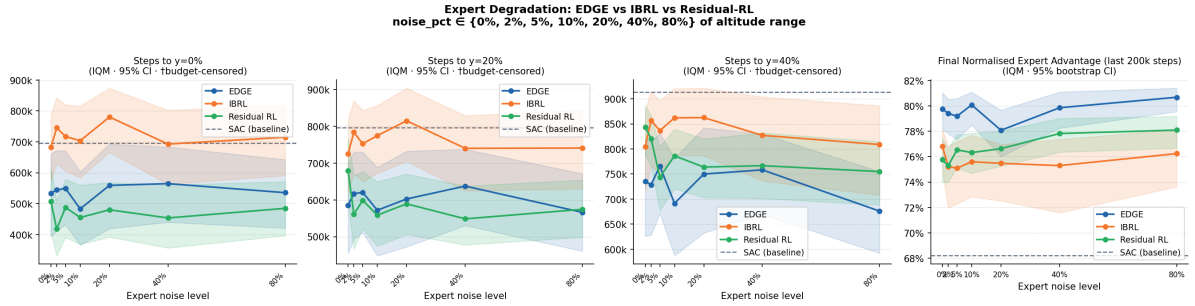



**Figure 8.8** – Expert robustness study: each panel shows a performance metric as a function of expert noise level $\rho \in \{0\%, 2\%, 5\%, 10\%, 20\%, 40\%, 80\%\}$. The first three panels report steps to reach NEA $\geq 0\%$, $20\%$, and $40\%$ respectively (lower is better; budget-censored runs included at the training budget); the rightmost panel reports the final NEA IQM over the last 200k steps (higher is better). Lines show IQM across 100 seeds; shaded bands are 95% bootstrap CI. The SAC baseline (dashed) is noise-free throughout.

In Figure 8.8, we can observe that EDGE and residual-RL maintain a clear advantage over the noise-free SAC baseline across all noise levels and all metrics. IBRL, however, offers little to no improvement over SAC for the first two sample efficiency thresholds ($NEA \geq 0\%$ and $NEA \geq 20\%$) even at $\rho = 0\%$, and its advantage erodes further as noise increases. For the final performance metric and the $40\%$ threshold, all three methods remain above SAC even at $\rho = 80\%$, where the expert's perceived target is almost entirely noise.

EDGE is the most robust of the three methods. Across all four panels its curves are the flattest, with final NEA remaining near $80\%$ and steps-to-threshold staying nearly constant as $\rho$ increases from $0\%$ to $80\%$. The decay schedule is the key mechanism: once $p_t = 0$, noisy expert actions stop entering the replay buffer entirely, so the asymptotic policy is never contaminated regardless of expert quality.

IBRL degrades most visibly in sample efficiency, with steps-to-threshold increasing with noise until they approach the SAC baseline at $\rho = 80\%$. This is consistent with the extrapolation error argument from Q3: argmax gating relies on Q-values to decide when to defer to the expert, but a noisy expert produces systematically biased Q-value estimates for its own actions, corrupting the gating signal. Without a decay mechanism, there is no way to shut off this corruption.

Residual RL shows mixed behavior. It is the fastest method to reach NEA$= 0\%$ across all noise levels, as the residual structure starts from the expert action and therefore trivially approaches the expert's performance, but it lags behind EDGE for the more demanding $40\%$ threshold and in final NEA.

A surprising result is that **performance does not degrade monotonically with noise**. For EDGE, final NEA is slightly higher at $\rho = 2$–$5\%$ than at $\rho = 0\%$; a similar non-monotonic pattern appears for residual-RL at medium noise levels. While the overlapping confidence intervals prevent strong conclusions, this hints at a counter-intuitive mechanism: a mildly noisy expert may enrich early exploration by injecting more diverse trajectories than a perfectly stable one, whose actions are by definition highly repetitive near the setpoint. Investigating whether deliberate, calibrated expert noise could be used as an exploration bonus is an interesting direction we leave for future work.

**Outtake** EDGE is robust to expert suboptimality. The decay mechanism provides a natural insulation as the expert shapes early exploration but cannot corrupt the asymptotic policy. In contrast, methods without decay degrade with noise because they have no mechanism to discount the expert's influence once it becomes unreliable. Crucially, no method is harmed by the expert relative to SAC, even at extreme noise levels, confirming that incorporating a stable expert is safe in practice.

**Industrial perspective**

The results presented in this chapter are encouraging for industrial deployment of EDGE. Four properties stand out. First, the degradation study (Section 8.5.4) shows that EDGE remains robust even when the expert is significantly corrupted: in industrial settings, the available controller is rarely a perfect expert, being instead a PID or MPC tuned to nominal conditions. The fact that EDGE maintains its advantage over SAC at high noise levels means it can be applied directly on top of whatever controller is already in place, without requiring a carefully calibrated expert. Second, the sample efficiency gains are practically meaningful: every training step in a real or high-fidelity simulation environment has a cost, and reaching target performance in roughly half the steps of the baseline translates directly into shorter deployment cycles. Third, the ablation studies show that EDGE is not sensitive to the exact values of its hyperparameters within a reasonable range, which reduces the tuning burden in settings where extensive hyperparameter search is impractical. Fourth, the expert-necessity ablation (Q4) confirms that the observed gains are driven by the expert's domain knowledge and not merely by the exploration schedule: replacing the expert with random actions yields performance at or below SAC, establishing that the method requires a meaningful expert to function. Taken together, these properties suggest that EDGE is a practical candidate for SPT problems in industry, where data is expensive, controllers already exist, and robustness to imperfection is non-negotiable.

## 8.6 Conclusion

**Position in the research landscape** This chapter addresses a gap identified in the related work: while SPT problems are paramount in industrial and robotic applications, no prior work had formalized them as a distinct RL problem class, nor proposed methods that explicitly exploit the structural advantage they provide, namely the availability of a stable partial expert. We introduced SP-MDPs as a formal framework for this class, proved that bias-optimal policies are necessarily stable, and showed that stable experts are therefore principled guides for exploration. Building on this, we proposed EDGE, a minimal modification to any off-policy RL algorithm that stochastically mixes expert and policy actions during data collection, with a decaying schedule that phases the expert out as the policy matures. Compared to IBRL, EDGE requires no critic modification and avoids the extrapolation error inherent in argmax gating, and achieves higher asymptotic performance (+17.0% IQM vs. SAC, against +12.7% for IBRL). Compared to residual-RL, it imposes no structural bias on the policy, allowing the agent to surpass the expert rather than being anchored to it, and again yields superior asymptotic performance (+17.0% vs. +11.1% IQM over SAC; see Table 8.1). The robustness study extends these contributions by characterizing how each method degrades under imperfect experts, a practically important question that neither the IBRL nor the residual-RL papers address.

**Limitations** The experimental validation is currently limited to the Plane task from TargetGym. This should be understood as a proof of concept: the task is directly representative of the industrial use case motivating this work, and the controlled setting allows clean ablations and statistical comparisons. Validation on a broader set of SP-MDP benchmarks is ongoing and will be reported in future work. Regarding the expert, EDGE requires one to be provided by the user, but this is a mild requirement in practice: most real-world SPT systems already operate under some existing controller, whether a PID, an MPC, or even a previously trained RL agent, any of which can directly serve as the expert without additional effort. The decay schedule requires knowing the total training budget $T$ upfront, which is standard practice in deep RL experiments and does not represent a meaningful constraint. The optimal decay fraction $f_{\text{decay}}$ benefits from tuning, but the Q1 ablation shows that performance is robust across a wide range of values, so a reasonable default suffices in most cases. Finally, the theoretical framework is developed in the average-reward setting, which is the natural formulation for SP-MDPs and provides the meaningful notions of gain- and bias-optimality used in the proofs. The experiments use discounted SAC, which is the standard in practice; while this gap is not bridged rigorously, the empirical results show that the method transfers cleanly to the discounted episodic setting, suggesting the theoretical insights are practically relevant beyond their formal scope.

**Future work / Open problems** Several directions are opened by the results in this chapter. First, the robustness study revealed a surprising non-monotonic pattern: mildly noisy experts can lead to slightly better performance than a perfect one, for both EDGE and residual-RL. While the confidence intervals prevent strong conclusions from the current experiments, the pattern is consistent enough to suggest a genuine phenomenon. A plausible explanation is that a slightly noisy expert diversifies early exploration by sampling trajectories that a perfectly stable, near-constant expert would never visit, providing richer coverage of the state space near the target region. If confirmed, this would suggest that **deliberate, calibrated expert noise** could be used as an exploration bonus in SP-MDPs, turning expert suboptimality from a liability into an asset. Second, the Design Choices section identified critic ensemble disagreement as a principled mechanism to bridge fixed mixing and value-gap gating: the gate could behave like a coin flip when the critic is uncertain and sharpen toward argmax as it calibrates, combining the reliability of stochastic mixing early in training with the precision of Q-value gating later. Third, extending EDGE to on-policy base algorithms is a natural next step. Fourth, while this chapter motivates and evaluates EDGE within the SP-MDP framework, the algorithm itself imposes no structural requirement on the task: it only requires an off-policy base algorithm and any policy that can serve as an expert, stable or not. Testing EDGE on standard continuous control benchmarks outside of the SP-MDP setting, such as MuJoCo locomotion tasks augmented with a partial expert, would clarify how much of the observed gain is specific to the near-maximum saturation regime of SPT tasks, and how much transfers to the broader family of expert-assisted

RL problems. Finally, evaluating EDGE on the glass furnace industrial use case that originally motivated this work remains the most important validation target.

**Takeaway** EDGE demonstrates that a minimal, structurally motivated modification to an off-policy RL algorithm can yield consistent gains in both sample efficiency ($-26.5\%$ steps to reach 20% NEA) and asymptotic performance ($+17.0\%$ IQM over SAC; see Table 8.1) on SPT tasks. The key insight is that a stable expert should **inform** exploration rather than dictate actions, and should be **phased out** once the policy has learned enough to outperform it. This design, justified by the theoretical results on SP-MDPs, produces a method that is simple to implement, robust to expert noise, and competitive with more invasive expert integration strategies. Because the algorithm itself places no structural requirement on the task beyond the availability of an off-policy base and a partial expert, it is a candidate for expert-assisted RL well beyond the SPT setting.

Chapter 9 closes the thesis by drawing together all contributions, reflecting on their limitations, and charting the path from the aircraft proxy studied here to the real float glass furnace that originally motivated this work.

# Chapter 9

# General Conclusion and Perspectives

> Da heul al labour emañ ar boued
> *"After doing your best, time to rest"* (*lit. "After work, comes food"*)
>
> *Breton Proverb*

## 9.1 Context and Objectives

This thesis sits at the intersection of industry and academia, motivated by a concrete challenge: improving industrial process control, exemplified by glass furnace temperature control. This dual context gives rise to two interwoven objectives. From the industrial side, the goal is to assess whether deep RL constitutes a viable alternative to traditional controllers such as PID and MPC. From the academic side, the goal is to advance RL methods to meet the specific demands of industrial settings. These objectives are not independent: the scientific contributions are the means by which the industrial question is answered. Concretely, we identified four gaps standing between standard RL and industrial deployment (sample efficiency, training robustness, benchmark tooling, and integration of classical controllers) and proposed solutions for each. The following sections detail the contributions made toward each of these directions, revisit the research questions posed in the introduction, and outline the limitations and perspectives that remain.

## 9.2 Contributions

Chapter 4 provides the literature review that contextualizes the original contributions summarized below; as a survey chapter, it does not constitute an independent research contribution.

Chapter 3 lays the foundation for all subsequent work by formalizing glass furnace temperature control as an RL problem. This contribution proceeds in two steps. First, we characterize

the furnace task through six key properties from an RL perspective: partial-observability, non-stationarity, real physics, setpoint tracking, stability around the setpoint, and delay. These properties both constrain the problem (e.g., partial-observability, delay) and can be leveraged to guide the search for good policies (e.g., setpoint tracking, stability around the setpoint). Second, given the prohibitive safety and cost requirements of direct interaction with a real furnace, and the absence of a viable physics simulator, we sought a proxy problem sharing these six properties. We identified airplane altitude control as a strong match: it naturally satisfies all six properties and enables tractable simulation based on aerodynamic equations. This choice is pragmatic: the airplane was simply the first system found to satisfy all six properties while remaining accessible in terms of simulation and required physics knowledge. Other systems sharing these properties could serve as proxies just as well; chemical reactor control, for example, is structurally compatible with this framework, and a standard simulation for it is already included in TargetGym via PC-gym. This result anchors the subsequent experimental chapters in a problem class that is structurally analogous to the industrial setting, sharing the six key properties identified above. Whether solutions found on the airplane proxy transfer to the actual furnace remains an open question and the primary validation target for future work.

Online RL is inherently affected by non-stationarity: because the policy changes continuously during training, the value network must track a moving target, which destabilizes training and limits its ability to adapt. Chapter 5 addresses this issue by introducing EVarEst, a method that offers explicit control over the bias–VRE (Variance of Residual Errors) weighting in the critic objective through a single hyperparameter $\alpha$. We showed that EVarEst generalizes both the standard MSE and AVEC objectives, which emerge as special cases. Augmenting SAC with EVarEst and a carefully selected $\alpha$ outperformed both baseline SAC and AVEC-SAC on 8 out of 9 MuJoCo and AdroitHand tasks (average improvement of $+13.29\%$ with per-task tuned $\alpha$); the same evaluation with PPO yielded an average improvement of $+28.7\%$ over baseline PPO on 7 out of 9 tasks. Together, these results demonstrate that finer control over the bias-variance trade-off in critic training translates into measurable performance gains when $\alpha$ is appropriately chosen. EVarEst was not validated on the industrial proxy task; its relevance to long-running systems such as the glass furnace, where adaptability to evolving conditions is critical, is a natural direction for future work.

Deep RL training is computationally demanding, and the industrial setting amplifies this in two ways: the tasks are less studied, requiring extensive experimentation to calibrate methods, and the physical systems are too costly to interact with directly, making fast simulation essential. Chapter 6 addresses this through an analytical contribution: a thorough study of parallelization in deep RL, covering both its algorithmic implications and its practical implementation. Beyond raw wall-clock speedup, the key insight is that parallelization reshapes the statistical properties of training data: collecting transitions from multiple independent environments simultaneously reduces temporal correlation and non-stationarity far more effectively than a single sequential

agent, which stabilizes gradient-based optimization. At scale, the diversity of environments also improves policy robustness and narrows the sim-to-real gap. On the implementation side, the chapter identifies JAX as the enabling technology for end-to-end GPU execution, eliminating the CPU-GPU transfer bottleneck that otherwise negates the gains from parallelization. The chapter also outlines a wall-clock-to-sample-efficient pipeline in which a policy is first pre-trained on thousands of parallel simulations, then fine-tuned on the real system, a regime already standard in robotics and directly applicable to industrial process control. This pipeline is proposed as a blueprint for future work rather than experimentally validated within this thesis.

Reliable and reproducible experimentation at scale requires dedicated software infrastructure. Chapter 7 presents three complementary tools built during this thesis. **ProbingEnvironments** is a suite of minimal diagnostic environments that isolate specific components of an actor-critic implementation, designed to enable principled integration testing of deep RL codebases; its adoption within this thesis itself was limited by the engineering overhead of integration, which illustrates the broader challenge of tooling adoption in the RL community. **AJAX** is a JAX-native RL library supporting both discounted-reward and average-reward algorithms, designed for end-to-end GPU execution and seed-parallel experimentation: its speedup over SB3 made the large-scale experiments of Chapter 8 tractable where a PyTorch-based approach was not. **TargetGym** completes the stack with a suite of JAX-native, physically grounded environments for setpoint tracking tasks, providing the benchmark against which the algorithms of this thesis are evaluated. These three tools are engineering contributions rather than scientific novelties: they constitute the infrastructure that makes the scientific work tractable and reproducible. Together, they embody the software engineering standards that rigorous RL research on industrial tasks requires.

Chapter 8 addresses the final gap: exploiting the structural properties of setpoint tracking tasks and the existence of pre-existing classical controllers to improve both sample efficiency and asymptotic performance. We first formalize setpoint tracking as a distinct RL problem class, SPT-MDPs, defined by a reward structure tied to proximity to a target setpoint. The framing is novel: we prove that the optimal policy in an SPT-MDP must itself converge to the setpoint, which implies that any expert sharing this convergence property (such as a stable classical controller) is a theoretically principled guide for exploration. Building on this, we introduce EDGE (Expert Decayed-Guided-Exploration), a minimal modification to any off-policy RL algorithm, that stochastically mixes expert and policy actions during data collection, with a decaying schedule that phases the expert out as the policy matures. Unlike residual-RL, EDGE imposes no structural bias on the policy, allowing it to surpass the expert rather than being anchored to it. Unlike IBRL, the decay schedule phases the expert out entirely, preventing permanent corruption of Q-value estimates by suboptimal expert transitions. Despite this simplicity, EDGE achieves statistically superior asymptotic performance over all baselines

($+17.0\%$ IQM over SAC; $p < 0.0091$; note that IBRL is represented here by its action-selection component only, not the full published method) and significantly better sample efficiency than SAC at intermediate performance thresholds ($-26.5\%$ steps to reach 20% NEA; $p < 0.0039$), confirmed by permutation tests with Holm–Bonferroni correction over 100 seeds; differences with IBRL and residual-RL on sample efficiency metrics did not reach statistical significance. A robustness study further shows that EDGE maintains its advantage over SAC across all tested levels of expert suboptimality, induced by corrupting the expert's target setpoint (up to 80% perturbation), and hints at a counter-intuitive pattern: mild expert noise may slightly improve performance, possibly because a slightly imperfect expert diversifies early exploration beyond what a perfectly stable, near-constant expert would visit; confidence intervals prevent strong conclusions, and this remains a conjecture to be confirmed in future work. This robustness result is arguably the most practically significant finding of this thesis. In real industrial settings, a perfect expert rarely exists: the available controller is typically a PID or MPC tuned to nominal conditions, which will be suboptimal under disturbances, wear, or recipe changes. The fact that EDGE retains its advantage over SAC even at extreme noise levels means it can be deployed directly on top of whatever controller is already in place, with no requirement for expert calibration or redesign. This removes the last major identified practical barrier specific to the expert-integration problem: the requirement for a well-calibrated expert. An ablation further confirms that the observed gains are driven by the expert's domain knowledge rather than by the decay schedule coupled to stochastic action selection alone: replacing the expert with a random-action policy yields performance at or below the SAC baseline, ruling out the explanation that the mechanism itself accounts for the improvement independently of expert quality. It should be noted that the experimental validation of EDGE currently rests on a single environment (the Plane task from TargetGym), which is representative of the industrial setting by construction but does not constitute broad empirical coverage; validation on a wider suite of SPT-MDP benchmarks is the immediate next step.

## 9.3 Research Questions Revisited

We now return to the five research questions posed in the introduction and state what this thesis can and cannot answer.

(**1**) **What structural properties characterize industrial process control as an RL problem, and how can they be used to rigorously justify the choice of a simulation proxy?** Industrial process control, as exemplified by glass furnace temperature control, is characterized by six structural properties from an RL perspective: partial-observability, non-stationarity, real physics, setpoint tracking, stability around the setpoint, and delay. These properties define the problem class and provide a principled criterion for evaluating candidate proxies. Airplane altitude control satisfies all six properties and admits a tractable simulation based on aerodynamic

equations, making it a structurally justified proxy for algorithmic research. Whether this structural equivalence is sufficient for solutions to transfer to the real furnace remains an open question.

**(2) How can value network training be made robust to the policy non-stationarity inherent to online RL?** Policy non-stationarity forces the value network to continuously adapt to a shifting target distribution, which degrades plasticity and leads to suboptimal value estimates. EVarEst addresses this by introducing a single hyperparameter $\alpha$ that controls the relative weighting of bias and VRE in the critic objective, with the standard MSE and the AVEC objectives as special cases at $\alpha = 0.5$ and $\alpha = 0$ respectively. Selecting $\alpha$ appropriately for each task yields consistent performance improvements, averaging $+13.29\%$ over SAC on 8 out of 9 MuJoCo and AdroitHand benchmarks with per-task tuned $\alpha$, and $+28.7\%$ over PPO on 7 out of 9 tasks. A principled method for setting $\alpha$ without per-task tuning remains an open problem.

**(3) How can parallelism make deep RL training tractable for industrial tasks, which require far more experience than standard benchmarks and whose less-studied nature demands extensive experimentation, and what further properties does it induce?** Parallelism reduces wall-clock training time by collecting experience from multiple independent environments simultaneously, making large-scale experimentation feasible on consumer hardware. Beyond speed, it reshapes the statistical properties of training: simultaneous collection from diverse environments reduces temporal correlation and non-stationarity in the replay buffer, which stabilizes gradient updates. At scale, exposure to a wide variety of initial conditions also improves policy robustness and narrows the sim-to-real gap. Practically, end-to-end GPU execution is necessary to realize these gains, as the CPU-GPU transfer bottleneck in conventional pipelines not only negates the speedup from parallelization but, as measured in this chapter, makes the hybrid CPU+GPU pipeline strictly slower than a pure CPU baseline. A wall-clock-to-sample-efficient pipeline exploiting these properties for industrial deployment is outlined as a blueprint, though not experimentally validated in this thesis.

**(4) What software infrastructure is needed for correct and reproducible RL research on industrial benchmarks?** Three complementary tools were identified as necessary and built during this thesis. Integration testing via **ProbingEnvironments** provides a principled way to verify that each component of an actor-critic algorithm functions correctly before running expensive experiments. A fast, reliable training library (**AJAX**) enables end-to-end GPU execution and seed-parallel experimentation, making the scale of experiments required for statistical rigor tractable. A domain-specific benchmark suite (**TargetGym**) provides physically grounded, hardware-accelerated environments for setpoint tracking, giving the community a common evaluation ground for this problem class. Together, these tools constitute the minimal infrastructure for rigorous RL research in this setting.

**(5) How can existing classical controllers be integrated into RL training to improve both sample efficiency and final performance on setpoint tracking tasks?** In setpoint tracking, a classical controller already operating near the target provides stable transitions that are informative early in training. EDGE exploits this by stochastically mixing expert and policy actions during data collection, with a decaying schedule that phases the expert out as the policy matures. This approach avoids the structural bias of residual RL (which anchors the policy to the expert) and the Q-value corruption of argmax gating without decay (as in IBRL). The result is statistically superior asymptotic performance over all baselines ($+17.0\%$ IQM over SAC; $p < 0.0091$; where IBRL is compared using its action-selection component only) and significantly better sample efficiency than SAC ($-26.5\%$ steps to reach 20% NEA; $p < 0.0039$), while remaining robust to expert suboptimality across all tested noise levels. The method requires only an off-policy base algorithm and any existing controller as expert, making it directly applicable to industrial settings where a classical controller is already deployed.

## 9.4 Limitations

**No experiment on the furnace or a furnace simulation** The most significant gap between the ambition of this thesis and its experimental scope is the absence of any experiment on the actual glass furnace or even a dedicated furnace simulation. All algorithmic validation was done on the airplane altitude control proxy, chosen for its structural similarity to the furnace problem. While this choice is principled, it leaves the industrial claim untested: we have shown that our methods work on a structurally analogous problem, not on the problem itself. Furthermore, even if a furnace simulation were available, the sim-to-real transfer question would remain open. The physical complexity of a glass furnace, including non-stationary batch composition, thermal wear, and sensor drift, means that a simulation accurate enough to support meaningful transfer is itself a substantial research and engineering effort. Validating the full pipeline on the real system is the ultimate open problem left by this thesis.

**The mechanism of EVarEst is not fully understood** While EVarEst consistently improves performance when $\alpha$ is tuned per task, the underlying reason why a particular value of $\alpha$ is optimal for a given environment remains unclear. No principled method exists for setting $\alpha$ without a hyperparameter search, which limits practical adoption in settings where such a search is costly. A mechanistic understanding of how environment properties (e.g., degree of non-stationarity, reward density, action space dimensionality) determine the optimal bias-variance trade-off would be necessary to turn EVarEst into a fully practical tool.

**Tooling maturity** **ProbingEnvironments** was not used during the development of the algorithms in this thesis, which is an honest reflection of the integration overhead it currently requires. The library exists and is functional, but the friction of adoption remains high enough

that even within this thesis it was not systematically applied, limiting its demonstrated value. **TargetGym**, while providing a physically grounded starting point, currently contains a small number of tasks. A single benchmark environment (the Plane task) was used for all algorithmic evaluation in Chapter 8, which is insufficient to establish the generality of the results across the broader class of SPT-MDPs.

**EDGE validation scope** The experimental validation of EDGE is limited to the Plane task from TargetGym. Although this environment is directly representative of the industrial use case, it constitutes a single data point and does not establish the method's generality across SPT-MDP benchmarks or continuous control tasks more broadly. The ablation study in Chapter 8 covers the main design choices of EDGE and confirms two important properties of the experimental setup: the TD-target bootstrap omitted from the IBRL comparison has no measurable effect on either method (see Appendix C.4), and the expert's domain knowledge is genuinely necessary, as replacing it with a random-action policy yields performance at or below SAC (see Section 8.5.3).

**Safety** This thesis does not address safe RL, constraint satisfaction, or the interaction of the proposed methods with safety requirements. Industrial process control is safety-critical: a glass furnace operating outside its thermal envelope risks equipment damage and energy waste. Any real deployment of RL-based controllers in industrial settings would require, at minimum, a formal treatment of operating constraints and a safe exploration strategy. This gap is orthogonal to the contributions made here but must be addressed before deployment.

## 9.5 Perspectives

**Furnace simulation and benchmark expansion** The most impactful next step is to develop a coarse but physically grounded simulation of the glass furnace and integrate it into TargetGym. This single effort addresses two limitations simultaneously: it provides the missing validation environment for the industrial use case, and it enriches TargetGym with a more complex task that better challenges the methods developed in this thesis. A furnace simulation does not need to be perfectly accurate to be useful; a model that captures the dominant dynamics (thermal inertia, non-stationarity from batch variation, actuation delay) would already constitute a meaningful step toward closing the gap between the airplane proxy and the real system. In the longer term, validating the full pipeline on the actual furnace remains the ultimate target.

**Mechanistic understanding and principled selection of $\alpha$ in EVarEst** Re-implementing EVarEst in JAX would provide a significantly faster and more flexible experimental platform, enabling the large-scale sweeps needed to characterize $\alpha$ as a function of environment properties. Chapter 5 includes post-hoc analyses that hint at which task characteristics correlate with the

optimal $\alpha$; a systematic study building on these observations could yield a predictive model or a simple heuristic for setting $\alpha$ without per-task search. This would close the gap between EVarEst as a research result and EVarEst as a deployable tool.

**ProbingEnvironments adoption** The primary barrier to adopting **ProbingEnvironments** is the effort required to write connectors between the test suite and a given RL codebase. Developing a broad library of such connectors, covering some of the most widely used frameworks (Stable-Baselines3, CleanRL), and AJAX, would dramatically lower this barrier. Automated connector generation, leveraging the structured interfaces of modern RL libraries, is a promising direction. A validated connector for AJAX in particular would close the loop: the algorithms developed in this thesis would themselves be subject to the principled testing infrastructure also contributed by this thesis.

**EDGE generalization** The key remaining experiment is to evaluate EDGE on the full suite of SPT-MDP tasks in TargetGym (including a future furnace simulation), which would establish whether the observed gains generalize across the problem class. The Plane task used throughout this thesis is representative by construction but remains a single point in a broad space of setpoint tracking problems. Validation on tasks with different dynamics, action spaces, and levels of difficulty would either solidify the claim that EDGE is a general method for SPT-MDPs or reveal the conditions under which it fails, both of which would be of value from both an academic and industrial perspective. The furnace simulation in particular would close the loop on the industrial motivation of this thesis: confirming that the gains observed on the airplane proxy transfer, even partially, to the original problem would constitute the most meaningful validation possible within the scope of this work.

**Safe exploration and constraint satisfaction** Before any of the methods developed in this thesis can be deployed on a real glass furnace, safety constraints must be addressed. Industrial process control operates under hard limits (temperature bounds, actuation limits, ramp rates) whose violation has direct physical and economic consequences. Integrating safe RL methods (such as constrained policy optimization or safety layers) with the EDGE framework is a necessary step toward real deployment.

**Closing Remarks** Deep RL has become mature to the point where industrial deployment is almost within reach. Landmark results in plasma control, chemical dosing, and robotic manipulation have established that the gap between academic benchmarks and real physical systems is closable. This thesis addresses that challenge directly: it identifies the structural properties that define industrial process control as an RL problem, proposes and validates methods that address its specific challenges, and provides the open-source infrastructure that makes this research accessible and reproducible. Given the global scale of industrial processes and their

direct contribution to energy consumption and greenhouse gas emissions, the potential impact of better control strategies is substantial and measurable. By providing standardized, physically grounded benchmarks through TargetGym, this thesis invites practitioners to evaluate their algorithms in this setting and push the field forward. The gap between academic RL and industrial deployment is real, but it is narrowing; this thesis contributes a principled step in that direction, grounded in the physical and operational realities of a concrete industrial problem.

# Appendix A

# Complements on Chapter 5

## A.1 Proof of lemmas

### A.1.1 Proof of lemma 5.1

We have:

$$\begin{aligned}
MSE(\hat{y}, y) &= \mathbb{E}\left[(\hat{y} - y)^2\right] \\
&= \mathbb{E}\left[(\hat{y} - y - \mathbb{E}[\hat{y} - y] + \mathbb{E}[\hat{y} - y])^2\right] \\
&\overset{(a)}{=} \mathbb{E}\left[(\hat{y} - y - \mathbb{E}[\hat{y} - y])^2\right] + \mathbb{E}[\hat{y} - y]^2 + 2\mathbb{E}[\hat{y} - y]\mathbb{E}\left[\hat{y} - y - \mathbb{E}[\hat{y} - y]\right] \\
&\overset{(b)}{=} \mathbb{E}\left[(\hat{y} - y - \mathbb{E}[\hat{y} - y])^2\right] + \mathbb{E}[\hat{y} - y]^2 \\
&= var\left[\hat{y} - y\right] + \mathbb{E}[\hat{y} - y]^2
\end{aligned}$$

Where $(a)$ follows from expansion of the square and $(b)$ by linearity of expectation showing that the last term is zero. Then, because the covariance is bilinear,

$$MSE(\hat{y}, y) = var\left[\hat{y}\right] + var\left[y\right] - 2Covar(y, \hat{y}) + Bias(\hat{y}, y)^2.$$

### A.1.2 Proof of equation (5.4)

We start from eq (5.3):

$$\begin{aligned}
\mathcal{L}(\alpha_{\text{ev}}) &= \alpha_{\text{ev}} Bias(f_\phi, \hat{Q}^\pi) + (1 - \alpha_{\text{ev}}) Var(f_\phi - \hat{Q}^\pi) \\
&= \alpha_{\text{ev}} Bias(f_\phi, \hat{Q}^\pi) + \alpha_{\text{ev}} Var(f_\phi - \hat{Q}^\pi) - \alpha_{\text{ev}} Var(f_\phi - \hat{Q}^\pi) \\
&\quad + (1 - \alpha_{\text{ev}}) Var(f_\phi - \hat{Q}^\pi)
\end{aligned}$$

$$
\begin{aligned}
&= \alpha_{\text{ev}} MSE(f_\phi, \hat{Q}^\pi) + (1 - 2\alpha_{\text{ev}}) Var(f_\phi - \hat{Q}^\pi) \\
\frac{\mathcal{L}(\alpha_{\text{ev}})}{\alpha_{\text{ev}}} &= MSE(f_\phi, \hat{Q}^\pi) + \frac{(1 - 2\alpha_{\text{ev}})}{\alpha_{\text{ev}}} Var(f_\phi - \hat{Q}^\pi) \\
\mathcal{L}_{EVarEst}(\alpha_{\text{ev}}) &= MSE(f_\phi, \hat{Q}^\pi) + \frac{(1 - 2\alpha_{\text{ev}})}{\alpha_{\text{ev}}} Var(f_\phi - \hat{Q}^\pi)
\end{aligned}
$$

## A.2 Additional figures

### A.2.1 Mean and variance of value network predictions

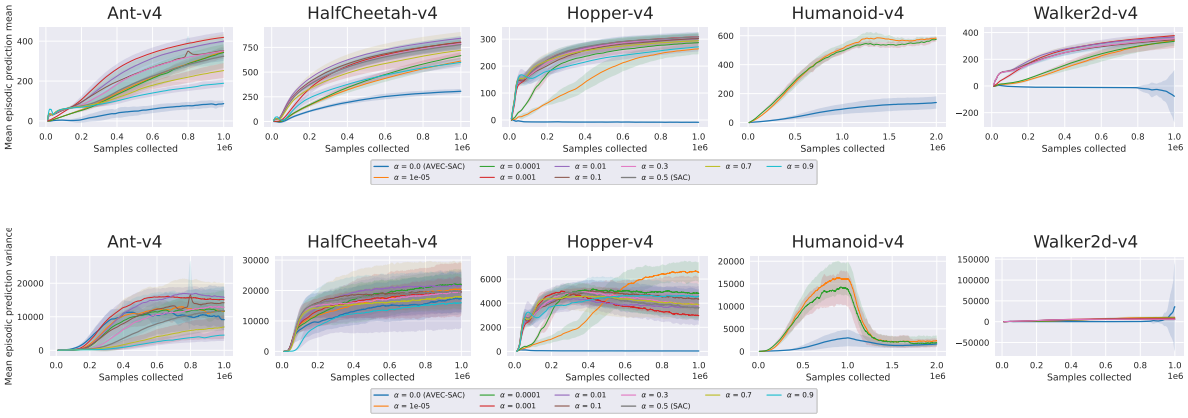



**Figure A.1** – EVarEst-SAC critic predictions evolution during training for different $\alpha_{\text{ev}}$ values (Top: prediction mean, Bottom: prediction variance)

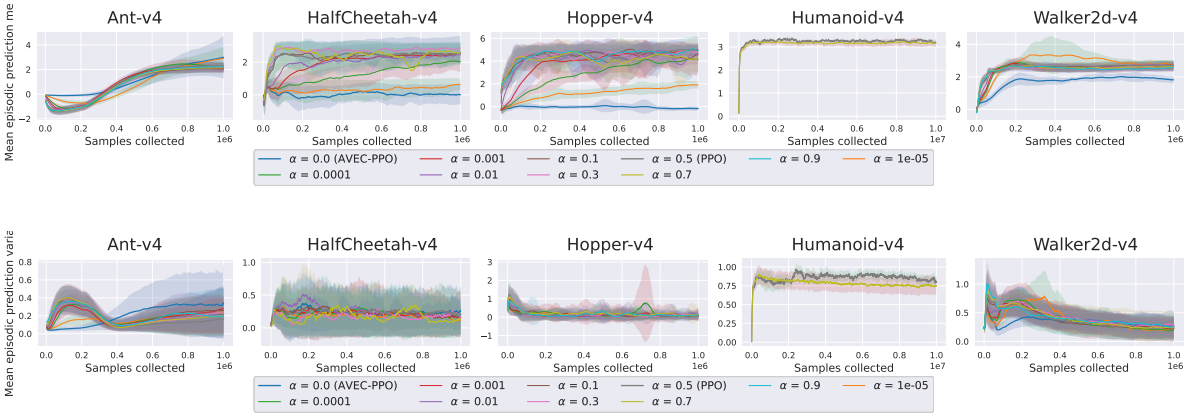



**Figure A.2** – EVarEst-PPO critic predictions evolution during training for different $\alpha_{\text{ev}}$ values (Top: prediction mean, Bottom: prediction variance)

### A.2.2 AdroitHandPen early termination

We take a deeper look at AdroitHandPen where the policy returns seem to be correlated to a high $\alpha_{\text{ev}}$ value instead of a low $\alpha_{\text{ev}}$ value like on MuJoCo. In this task the agent controls a

robotic hand that has to manipulate a pen to match a given orientation and position. Looking at the specificities of this task we notice that although it has a fixed episode length like all other AdroitHand tasks, it is possible for the hand to drop the pen. If this happens the agent suffers from a constant negative reward value without being able to pick up the pen, this leads the value estimator to try and learn from a signal that is independent from its actions. Our hypothesis is that this is the property that separates it from other tasks in our benchmark. To verify this we compare the results of $EVarEst$-SAC on AdroitHandPen and AdroitHandPen with early termination where the episode ends if the pen drops. The agent is train the agent on this variation, while it still evaluated on the fixed length task for comparison.

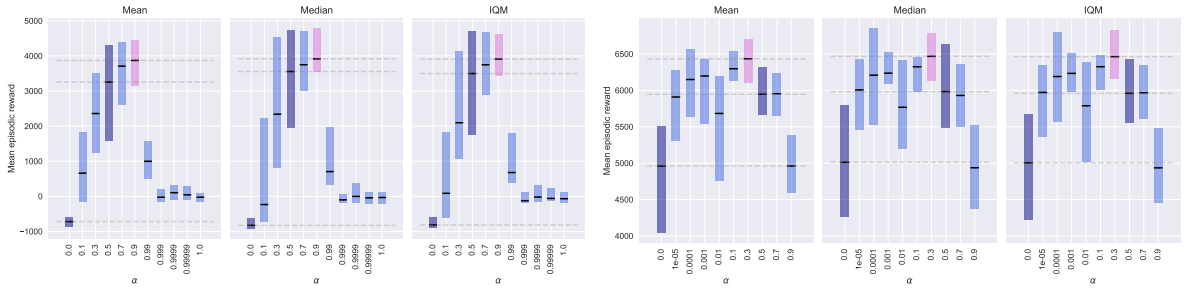



**Figure A.3** – left without early termination, right with early termination.

In Figure A.3 we can observe that by adding early termination we observe a relation between $\alpha_{\text{ev}}$ and policy performance much closer to what we observed on MuJoCo. We also notice that on the normal task without early termination, although putting more weight on bias seems to be correlated with better performance, it has a limit where $\alpha_{\text{ev}} > 0.9$ leads an important drop in performance.

Our hypothesis about these results is that if the bias is to be more important than the variance of the residual errors, it likely means that the variance of the residual errors is too hard to reduce and spending representation capacity on this aspect of the MSE is a bad investment. The possibility for the pen to drop and have negative rewards there onward leads to an important variance in episodic returns that is hard for the model to estimate. Therefore avoiding trying to estimate it to some degree seems to be beneficial as it is almost noise-like from the estimator perspective.

### A.2.3 PPO Critic performance




**Figure A.4** – Relative difference between $EVarEst$-PPO and PPO baseline for value estimation metrics (MSE, Bias$^2$ and VRE) averaged on the last $10^5$ timesteps with different $\alpha_{\text{ev}}$ values.

### A.2.4 PPO Feature rank

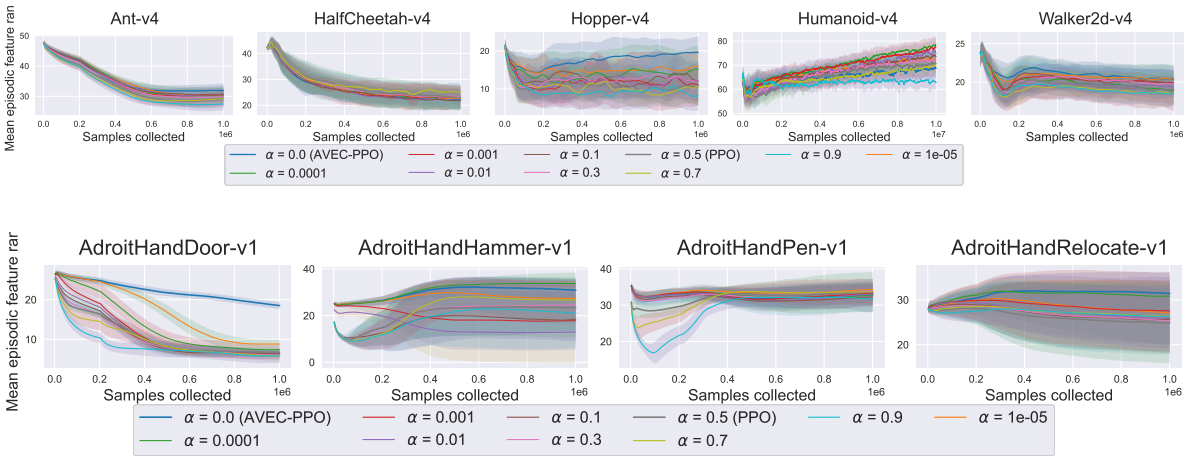



**Figure A.5** – Feature rank of $EVarEst$-PPO evolution during training (Top: Mujoco, Bottom: AdroitHand).

### A.2.5 Robustness on AdroitHand

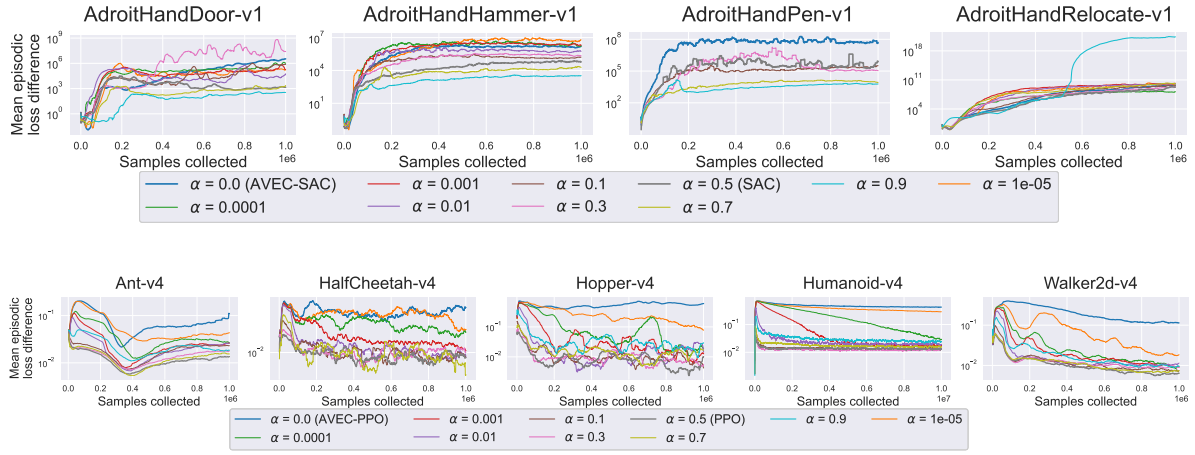



**Figure A.6** – Relative difference between $EVarEst$ and baseline MSE for value loss over training for different $\alpha_{\text{ev}}$ values on AdroitHand tasks (Top: SAC, Bottom: PPO).

### A.2.6 Performance during training

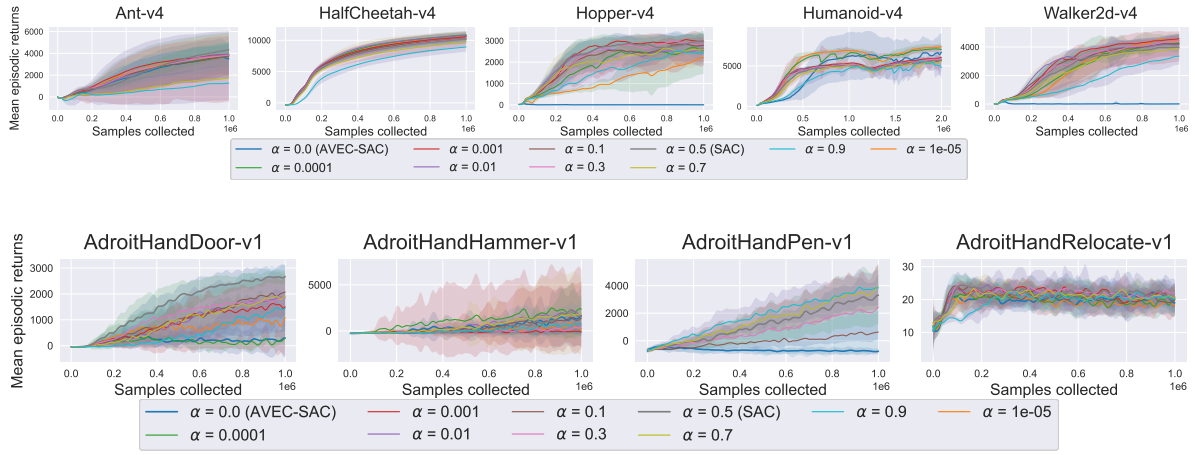



**Figure A.7** – Mean episodic return of $EVarEst$-SAC during training for different $\alpha_{\text{ev}}$ values(Top: Mujoco tasks, Bottom: AdroitHand tasks).

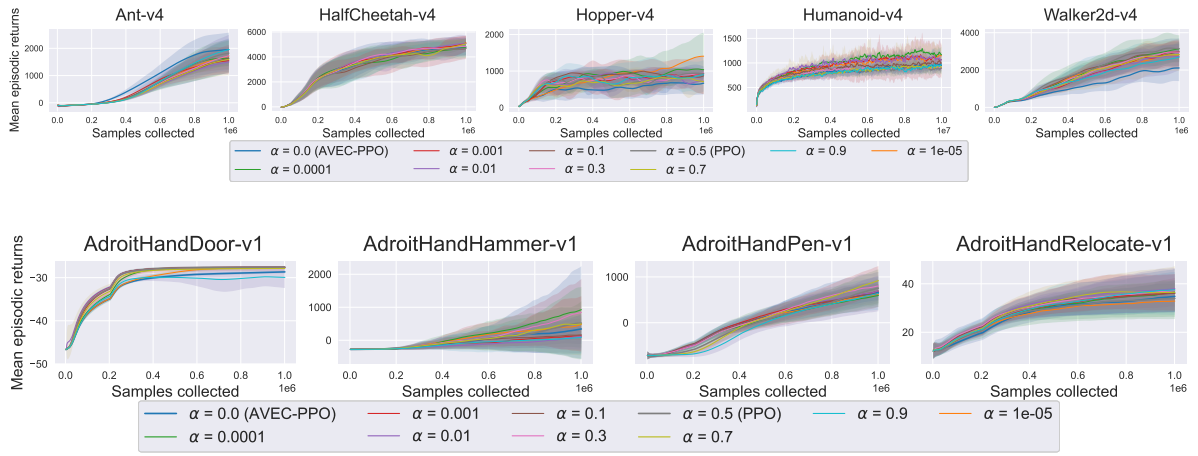



**Figure A.8** – Mean episodic return of $EVarEst$-PPO during training for different $\alpha_{\text{ev}}$ values(Top: Mujoco tasks, Bottom: AdroitHand tasks).

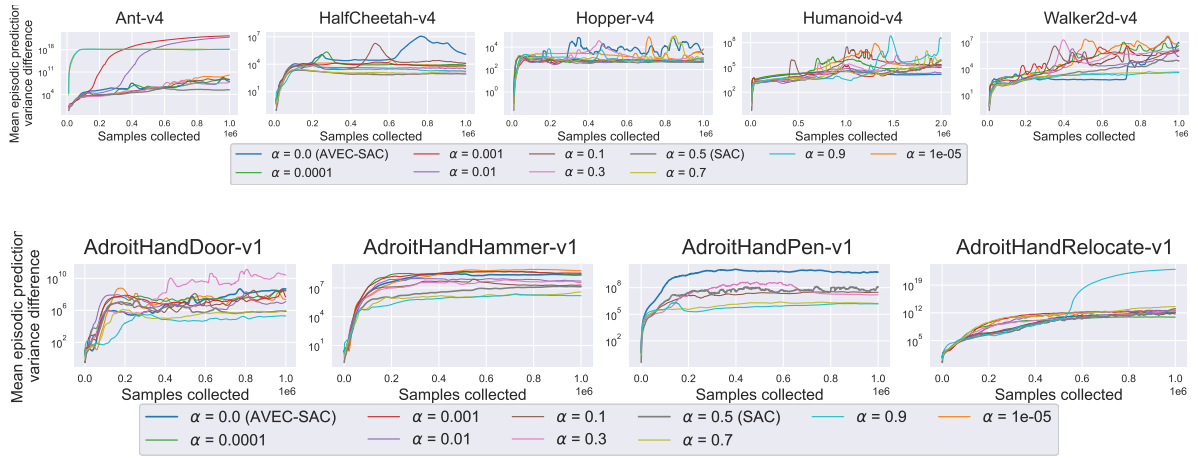



**Figure A.9** – Mean episodic length of $EVarEst$-SAC during training for different $\alpha_{\text{ev}}$ values(Top: Mujoco tasks, Bottom: AdroitHand tasks).

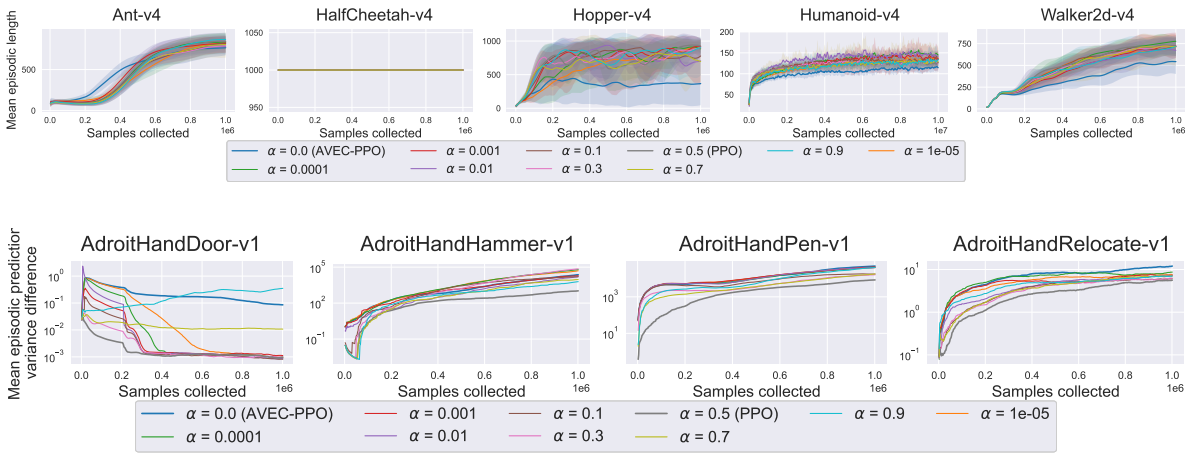



**Figure A.10** – Mean episodic length of $EVarEst$-PPO during training for different $\alpha_{\text{ev}}$ values(Top: Mujoco tasks, Bottom: AdroitHand tasks).

## A.3 Critic comparison protocol

In order to provide meaningful comparison between value networks, we argue that they should be trained and evaluated on the same data. The data should be representative of our setting to provide insightful results. As a consequence we chose to use policies obtained from the corresponding algorithms to generate our data (states visited, actions taken during rollout of the policy, and value surrogates of the policy).

In order to reduce experimental load, we achieve this during the training of the agent by training an additional value network, disconnected from the policy. Both networks function like a standard value network, observing the data generated by the actor, estimating its value, and learning from the policy returns; however only the first critic is used for policy updates.

## A.4 Hyperparameters

In Table A.1 and A.3 we report the list of hyperparameters common to all continuous control experiments. All environments use the default hyperparameters unless specified otherwise in Tables A.2 and A.4. Hyperparameters are taken from stable-baselines3 (Raffin et al., 2021) default parameters, and rl-zoo3 (Raffin, 2020) when they differ from the default stable-baselines3 value. rl-zoo3 provides optimized hyperparameters for different agents and environments. Note that the parameters were optimized for the original algorithms (PPO and SAC) and not for their modified versions ($EVarEst$-PPO and $EVarEst$-SAC). Hence they should favor the original algorithms.

**Table A.1** – Default hyperparameters for both SAC and $EVarEst$-SAC.

| Parameter | Value |
|---|---|
| Number of training steps | $10^6$ |
| Adam stepsize | $3 \cdot 10^{-4}$ |
| Discount ($\gamma$) | 0.99 |
| Replay buffer size | $10^6$ |
| Batch size | 256 |
| Nb. hidden layers | 2 |
| Nb. hidden units per layer | 256 |
| Nonlinearity | ReLU |
| Target smoothing coefficient ($\tau$) | 0.005 |
| Target update interval | 1 |
| Gradient steps | 1 |
| Learning starts | $10^4$ |

**Table A.2** – Environment specific hyperparameters for SAC and $EVarEst$-SAC.

| Environment | Parameter | Value |
|---|---|---|
| Humanoid-v4 | Number of training steps | $2 \cdot 10^6$ |

**Table A.3** – Default hyperparameters for both PPO and $EVarEst$-PPO.

| Parameter | Value |
|---|---|
| Number of training steps | $10^6$ |
| Horizon ($T$) | 2048 |
| Adam stepsize | $3 \cdot 10^{-4}$ |
| Batch size | 64 |
| Nb. epochs | 10 |
| Nb. hidden layers | 2 |
| Nb. hidden units per layer | 64 |
| Nonlinearity | tanh |
| Discount ($\gamma$) | 0.99 |
| GAE parameter ($\lambda$) | 0.95 |
| Clipping parameter ($\varepsilon$) | 0.2 |
| Maximum gradient norm | 0.5 |
| State and reward normalization | True |
| Nb. environments | 1 |
| Value function loss coefficient | 0.5 |
| Initialization log standard deviation | 0.0 |
| Orthogonal initialization | True |

**Table A.4** – Environment specific hyperparameters for PPO and $EVarEst$-PPO.

| Environment | Parameter | Value |
|---|---|---|
| HalfCheetah-v4 | Discount ($\gamma$) | 0.98 |
| | Horizon ($T$) | 512 |
| | Adam stepsize | $2.0633 \cdot 10^{-5}$ |
| | Entropy coefficient | 0.000401762 |
| | Clipping parameter ($\varepsilon$) | 0.1 |

(*continued*)

| Environment | Parameter | Value |
|---|---|---|
| | Nb. epochs | 20 |
| | GAE parameter ($\lambda$) | 0.92 |
| | Maximum gradient norm | 0.8 |
| | Value function loss coefficient | 0.58096 |
| | Initialization log standard deviation | -2.0 |
| | Orthogonal initialization | False |
| | Nb. hidden units per layer | 256 |
| | Nonlinearity | ReLU |
| Hopper-v4 | Discount ($\gamma$) | 0.999 |
| | Horizon ($T$) | 512 |
| | Adam stepsize | $9.80828 \cdot 10^{-5}$ |
| | Batch size | 32 |
| | Entropy coefficient | 0.00229519 |
| | Nb. epochs | 5 |
| | GAE parameter ($\lambda$) | 0.99 |
| | Maximum gradient norm | 0.7 |
| | Value function loss coefficient | 0.835671 |
| | Initialization log standard deviation | -2.0 |
| | Orthogonal initialization | False |
| | Nb. hidden units per layer | 256 |
| | Nonlinearity | ReLU |
| Humanoid-v4 | Number of training steps | $10^7$ |
| | Discount ($\gamma$) | 0.95 |
| | Horizon ($T$) | 512 |
| | Adam stepsize | $3.56987 \cdot 10^{-5}$ |
| | Batch size | 256 |
| | Entropy coefficient | 0.00229519 |
| | Clipping parameter ($\varepsilon$) | 0.3 |
| | Nb. epochs | 5 |
| | GAE parameter ($\lambda$) | 0.9 |
| | Maximum gradient norm | 2.0 |
| | Value function loss coefficient | 0.431892 |
| | Initialization log standard deviation | -2.0 |
| | Orthogonal initialization | False |
| | Nb. hidden units per layer | 256 |
| | Nonlinearity | ReLU |
| Walker2d-v4 | Discount ($\gamma$) | 0.99 |
| | Horizon ($T$) | 512 |
| | Adam stepsize | $5.05041 \cdot 10^{-5}$ |
| | Batch size | 32 |
| | Entropy coefficient | 0.00229519 |
| | Clipping parameter ($\varepsilon$) | 0.1 |
| | Nb. epochs | 20 |
| | GAE parameter ($\lambda$) | 0.95 |
| | Maximum gradient norm | 1 |
| | Value function loss coefficient | 0.871923 |

# Appendix B

# Longitudinal Flight Dynamics Model

## B.1 Introduction

In this appendix, we document the mathematical model underlying the `target_gym` plane environment used in this thesis. This environment simulates the *longitudinal* flight dynamics of a narrow-body commercial aircraft in a 2-D vertical plane, and serves as the testbed for the RL control policies studied in the main body. We make every computation explicit so that readers without an aeronautics background can follow the derivation. We also justify all numerical parameter values with references to the open literature and publicly available data for the Airbus A320 family, and we document the approximations that were deliberately made to keep the model tractable for RL training.

We base the reference aircraft on the Airbus A320-200 equipped with CFM56-5B engines (Airbus S.A.S., 2018; Jackson, n.d.).

## B.2 Coordinate System and State Variables

### B.2.1 Coordinate System

We restrict the simulation to the *longitudinal plane*: a 2-D Cartesian frame with horizontal axis $x$ (positive forward) and vertical axis $z$ (positive upward). No lateral or roll dynamics are modeled.

### B.2.2 State Variables

The state of the aircraft at time $t$ is described by seven scalar quantities plus two actuator states, listed in Table B.1.

Two derived kinematic quantities appear throughout:

**Table B.1** – State variables of the simulation.

| Symbol | Physical meaning | Unit |
|---|---|---|
| $x$ | Horizontal position | m |
| $z$ | Altitude (vertical position) | m |
| $\dot{x}$ | Horizontal velocity | $\mathrm{m\,s^{-1}}$ |
| $\dot{z}$ | Vertical velocity | $\mathrm{m\,s^{-1}}$ |
| $\theta$ | Pitch angle (body axis vs. horizon) | rad |
| $\dot{\theta}$ | Pitch rate | $\mathrm{rad\,s^{-1}}$ |
| $m$ | Aircraft mass | kg |
| $\delta_p$ | Throttle setting (actuator state) | dimensionless (0–1) |
| $\delta_s$ | Elevator deflection (actuator state) | rad |

$$\gamma = \arctan\left(\frac{\dot{z}}{\dot{x}}\right), \quad \text{flight-path angle: direction of the velocity vector w.r.t. horizon,} \tag{B.1}$$

$$\alpha = \theta - \gamma, \quad \text{angle of attack (AoA): angle between the body axis and the velocity vector.} \tag{B.2}$$

Figure B.1 illustrates the geometric relationship between $\theta$, $\gamma$, and $\alpha$.

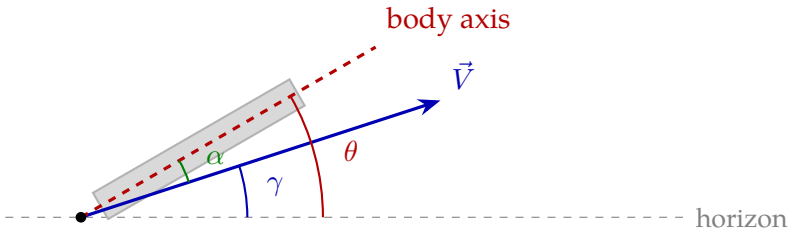



**Figure B.1** – Relationship between pitch angle $\theta$ (body axis relative to horizon), flight-path angle $\gamma$ (velocity vector relative to horizon), and angle of attack $\alpha = \theta - \gamma$ (body axis relative to velocity vector). All angles are positive in the nose-up/climb direction.

## B.3 Atmospheric Model

### B.3.1 Air Density

We use the *International Standard Atmosphere* (ISA) (International Organization for Standardization, 1975) troposphere model, valid from sea level to 11 km. Temperature decreases linearly with altitude:

$$T(z) = T_0 - L\,z, \qquad T_0 = 288.15\,\mathrm{K}, \quad L = 0.0065\,\mathrm{K\,m^{-1}}.$$

Pressure follows from the hydrostatic equation combined with the ideal-gas law:

$$P(z) = P_0 \left(\frac{T(z)}{T_0}\right)^{\frac{g}{R\,L}}, \qquad P_0 = 101\,325\,\mathrm{Pa}, \quad g = 9.806\,65\,\mathrm{m/s^2}, \quad R = 287.05\,\mathrm{J\,kg^{-1}\,K^{-1}}.$$

Air density is then obtained from the ideal-gas law:

$$\rho(z) = \frac{P(z)}{R\,T(z)}. \tag{B.3}$$

At sea level, Eq. (B.3) gives $\rho_0 = 1.225\,\mathrm{kg/m^3}$, in agreement with the ISA standard. Above 11 km, temperature is capped at 216.65 K; the episode altitude range (3000 m to 12 200 m) remains within the troposphere for all practical purposes.

### B.3.2 Speed of Sound

The local speed of sound in air is:

$$a(z) = \sqrt{\gamma_{\text{air}}\, R\, T(z)}, \qquad \gamma_{\text{air}} = 1.4,$$

yielding $a_0 = 340.3\,\mathrm{m\,s^{-1}}$ at sea level.

### B.3.3 Mach Number

Given total airspeed $V = \sqrt{\dot{x}^2 + \dot{z}^2}$, the Mach number is:

$$M = \frac{V}{a(z)}.$$

## B.4 Aerodynamic Forces

We decompose aerodynamic forces into *lift* (perpendicular to the velocity vector) and *drag* (opposing the velocity vector). Both are computed from the standard aerodynamic formula:

$$F = \tfrac{1}{2}\,\rho\, S\, C\, V^2, \tag{B.4}$$

where $S$ is the relevant reference area, $C$ is the dimensionless aerodynamic coefficient, and $V$ is total airspeed. Equation (B.4) is applied independently to each lifting surface: main wings, horizontal stabilizer, and elevator.

### B.4.1 Aerodynamic Coefficients

#### Lift Coefficient

We model the lift coefficient as a linear function of AoA with a smooth stall transition:

$$C_L^{\text{raw}}(\alpha) = \frac{C_{L,0} + C_{L,\alpha}\,\alpha_{\deg}}{1 + \exp\!\left[(\alpha_{\deg} - \alpha_{\text{stall}}) \cdot 1.5\right]},$$

where $\alpha_{\deg}$ is the angle of attack in degrees, $C_{L,0}$ is the zero-AoA lift coefficient, $C_{L,\alpha}$ is the lift-curve slope ($\mathrm{deg^{-1}}$), and $\alpha_{\text{stall}}$ is the stall angle. The logistic term in the denominator provides a

smooth, differentiable transition to the post-stall regime, replacing an abrupt discontinuity that would impede gradient-based optimization. The lift coefficient is then capped at a maximum value:

$$C_L(\alpha) = \min(C_L^{\text{raw}}(\alpha),\ C_{L,\max})\,.$$

**Parameter values.** For a typical transonic swept wing of the A320 class, the lift-curve slope is approximately $C_{L,\alpha} \approx 0.08$–$0.11\ \text{deg}^{-1}$ (Raymer, 2018; Torenbeek, 1982). The value $C_{L,\alpha} = 0.04\ \text{deg}^{-1}$ used here is conservative, partly accounting for the 2-D simplification of the model (no explicit finite-span correction is applied). The stall angle $\alpha_{\text{stall}} = 15°$ and maximum lift $C_{L,\max} = 1.5$ are representative of an A320 in clean configuration at moderate Mach numbers (Airbus S.A.S., 2017).

### Drag Coefficient

The drag polar relates total drag to lift through the zero-lift drag coefficient $C_{D,0}$ (profile and interference drag) and an induced drag factor $k$:

$$C_D(\alpha) = C_{D,0} + k\,C_L(\alpha)^2. \tag{B.5}$$

This *parabolic polar* is the standard first-order model used in preliminary aircraft design (Raymer, 2018; Torenbeek, 1982). The second term represents *induced drag*: energy dissipated in trailing vortices generated as a by-product of lift. For an elliptically loaded wing of aspect ratio $AR$, $k = 1/(\pi\, e\, AR)$, where $e \leq 1$ is the Oswald efficiency factor.

**Parameter values.** $C_{D,0} = 0.02$ and $k = 0.045$ are typical for a narrow-body jet at cruise: $C_{D,0}$ accounts for skin friction and form drag, while $k$ corresponds to an aspect ratio $AR \approx 9$ with efficiency $e \approx 0.8$, consistent with the A320 wing geometry (Airbus S.A.S., 2018).

### Mach-Number Corrections

Below the critical Mach number $M_{\text{crit}}$, compressibility amplifies both lift and drag. The *Prandtl–Glauert rule* (J. D. Anderson, 2016) gives:

$$\beta = \sqrt{1 - M^2}, \qquad C_L^{\text{comp}} = \frac{C_L}{\beta}. \tag{B.6}$$

Above $M_{\text{crit}}$, shock waves form on the wing surface, causing a rapid *drag rise*. This is captured by a quadratic wave-drag penalty:

$$C_D^{\text{comp}} = C_D + k_{\text{drag}}\,(M - M_{\text{crit}})^2 \qquad \text{if } M > M_{\text{crit}}. \tag{B.7}$$

**Parameter values.** $M_{\text{crit}} = 0.80$ is consistent with the A320 design cruise of $M = 0.78$ and published aerodynamic data (Airbus S.A.S., 2018). The coefficient $k_{\text{drag}} = 5.0$ was tuned to produce a qualitatively correct divergence drag curve.

## B.5 Lifting Surfaces and Pitch Moment Balance

### B.5.1 Main Wings

The main wings have a reference area $S_w = 122.6\,\text{m}^2$, matching the A320 wing planform area (Airbus S.A.S., 2018). We evaluate aerodynamic coefficients at the current angle of attack $\alpha$ and Mach number $M$ using the model of Section B.4.1:

$$L_w = \tfrac{1}{2}\,\rho\,S_w\,C_L(\alpha, M)\,V^2,$$
$$D_w = \tfrac{1}{2}\,\rho\,S_w\,C_D(\alpha, M)\,V^2.$$

The wings produce a pitching moment about the center of mass:

$$M_w = L_w \cdot l_w, \qquad l_w = 1.5\,\text{m},$$

where $l_w$ is the moment arm between the wing aerodynamic center and the aircraft center of mass. A positive $M_w$ induces a nose-up tendency.

### B.5.2 Horizontal Stabiliser

The horizontal stabilizer provides pitch stability. It operates at an effective AoA that is reduced by a fixed incidence angle of $3^\circ$ relative to the main wing:

$$\alpha_s = \alpha_{\text{deg}} - 3^\circ.$$

This offset approximates the trimmed-flight setting of the A320 horizontal stabilizer. The resulting forces and moment are:

$$L_s = \tfrac{1}{2}\,\rho\,S_s\,C_L(\alpha_s, M)\,V^2, \tag{B.8}$$

$$D_s = \tfrac{1}{2}\,\rho\, S_s\, C_D(\alpha_s, M)\, V^2, \tag{B.9}$$

$$M_s = -(L_s - D_s)\cdot l_t, \tag{B.10}$$

with $S_s = 27\,\mathrm{m}^2$ (estimated horizontal tail area (Torenbeek, 1982)) and $l_t = 15\,\mathrm{m}$ (tail moment arm). The negative sign in Eq. (B.10) reflects the stabilizing nature of the tail: a positive upward force at the tail produces a nose-down (restoring) pitching moment.

### B.5.3 Elevator

The elevator is a moveable portion of the horizontal tail. It is deflected by the agent-commanded elevator angle $\delta_s$. Its effective AoA combines the stabilizer incidence offset and the deflection:

$$\alpha_e = \alpha_{\mathrm{deg}} - \delta_{s,\mathrm{deg}} - 3^\circ.$$

We compute aerodynamic forces at $\alpha_e$ over area $S_e = 10\,\mathrm{m}^2$, then project them onto the longitudinal axis to account for the surface deflection:

$$F_e = L_e \cos\delta_s - D_e \sin\delta_s,$$
$$M_e = -F_e \cdot l_t.$$

### B.5.4 Total Pitching Moment and Pitch Equation

The net pitching moment is:

$$M_y = M_w + M_s + M_e.$$

Angular acceleration about the pitch axis follows from Euler's equation for rotation:

$$\ddot{\theta} = \frac{M_y}{I_y}, \tag{B.11}$$

where $I_y = 9 \times 10^6\,\mathrm{kgm}^2$ is the pitch moment of inertia, consistent with values reported in the literature for A320-class aircraft (Roskam, 2001; Federal Aviation Administration, 2004).

## B.6 Thrust Model

We model the total thrust produced by both engines as:

$$T = \delta_p \cdot T_{\text{SL}} \cdot \sigma(z) \cdot f_{\text{ram}}(M) \cdot f_{\text{shock}}(M),$$

where each factor is described below.

$T_{\text{SL}} = \mathbf{240\,000\,N}$ is the combined sea-level static thrust of two CFM56-5B engines (CFM International, 2010) ($2 \times 120\,\text{kN}$).

$\sigma(z) = \rho(z)/\rho_0$ is the density ratio. Since thrust is proportional to air mass flow through the engine, and mass flow scales with density, thrust decreases roughly in proportion to $\sigma$.

$f_{\text{ram}}(M) = \dfrac{1}{1 + k_1 M^2}$, $k_1 = 0.5$ captures the *ram drag* effect: at high Mach numbers, momentum is imparted to the airflow captured in the intake, which partially offsets the net forward thrust.

$f_{\text{shock}}(M) = \exp\!\left[-k_2 \max(M - M_{\text{crit}}, 0)^2\right]$, $k_2 = 10$ models the sharp thrust decrease that occurs when shocks form in the intake above $M_{\text{crit}} = 0.85$, reducing the aerodynamic efficiency of the inlet.

## B.7 Equations of Motion

### B.7.1 Force Decomposition

We express all forces in the world (inertial) Cartesian frame. Defining:

$$\hat{v} = \begin{pmatrix} \cos\gamma \\ \\ \sin\gamma \end{pmatrix}, \quad \mathbf{n} = \begin{pmatrix} -\sin\gamma \\ \\ \cos\gamma \end{pmatrix}, \quad \hat{b} = \begin{pmatrix} \cos\theta \\ \\ \sin\theta \end{pmatrix},$$

as the unit velocity vector, its perpendicular (aerodynamic lift direction), and the unit body-axis vector, the four forces acting on the aircraft are:

$$\begin{aligned} \mathbf{F}_{\text{drag}} &= -D_{\text{tot}}\,\hat{v}, \\ \mathbf{F}_{\text{lift}} &= L_{\text{tot}}\,\mathbf{n}, \\ \mathbf{F}_{\text{thrust}} &= T\,\hat{b}, \\ \mathbf{F}_{\text{weight}} &= -m\,g\,\hat{e}_z. \end{aligned}$$

where $L_{\text{tot}} = L_w + L_s + L_e$ and $D_{\text{tot}} = D_w + D_s + D_e$ are the total aerodynamic lift and drag.

### B.7.2 Translational Equations

Projecting Newton's second law $m\,\ddot{\mathbf{r}} = \sum \mathbf{F}$ onto the horizontal and vertical axes gives:

$$m\,\ddot{x} = T\cos\theta - D_{\text{tot}}\cos\gamma - L_{\text{tot}}\sin\gamma, \tag{B.12}$$

$$m\,\ddot{z} = T\sin\theta + L_{\text{tot}}\cos\gamma - D_{\text{tot}}\sin\gamma - mg. \tag{B.13}$$

### B.7.3 Full ODE System

Collecting the translational (Eqs. B.12–B.13) and rotational (Eq. B.11) equations, the complete state derivative is:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} x \\ \dot{x} \\ z \\ \dot{z} \\ \theta \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \dot{x} \\ \dfrac{T\cos\theta - D_{\text{tot}}\cos\gamma - L_{\text{tot}}\sin\gamma}{m} \\ \dot{z} \\ \dfrac{T\sin\theta + L_{\text{tot}}\cos\gamma - D_{\text{tot}}\sin\gamma - mg}{m} \\ \dot{\theta} \\ \dfrac{M_y}{I_y} \end{pmatrix}. \tag{B.14}$$

## B.8 Actuator Dynamics

Real aircraft actuators do not respond instantaneously to commands. Both the throttle and the elevator are modeled as *first-order lags*:

$$\frac{\mathrm{d}\delta_p}{\mathrm{d}t} = k_p\left(\delta_p^{\text{cmd}} - \delta_p\right),$$

$$\frac{\mathrm{d}\delta_s}{\mathrm{d}t} = k_s\left(\delta_s^{\text{cmd}} - \delta_s\right).$$

with $k_p = 0.05\ \mathrm{s}^{-1}$ and $k_s = 0.9\ \mathrm{s}^{-1}$. The much smaller value of $k_p$ reflects the slower spooling dynamics of a turbofan engine (several seconds to change thrust significantly) compared to the aerodynamic response of a control surface (sub-second). The discrete update for step $\Delta t$ is:

$$\delta_p^{(t+1)} = \delta_p^{(t)} + k_p\,\Delta t\,(\delta_p^{\mathrm{cmd},(t)} - \delta_p^{(t)}),$$
$$\delta_s^{(t+1)} = \delta_s^{(t)} + k_s\,\Delta t\,(\delta_s^{\mathrm{cmd},(t)} - \delta_s^{(t)}).$$

## B.9 Numerical Integration

We integrate the ODE system (B.14) numerically using the classical Runge–Kutta method of order 4 (RK4) (Hairer, Nørsett, and Wanner, 1993). Given state $(v_n, p_n)$ at time $t_n$ (where $v$ collects $[\dot{x}, \dot{z}, \dot{\theta}]$ and $p$ collects $[x, z, \theta]$), and a step size $h$, the scheme is:

$$\begin{aligned}
k_1 &= a(v_n,\ p_n), & \ell_1 &= v_n,\\
k_2 &= a\Big(v_n + \tfrac{h}{2}k_1,\ p_n + \tfrac{h}{2}\ell_1\Big), & \ell_2 &= v_n + \tfrac{h}{2}k_1,\\
k_3 &= a\Big(v_n + \tfrac{h}{2}k_2,\ p_n + \tfrac{h}{2}\ell_2\Big), & \ell_3 &= v_n + \tfrac{h}{2}k_2,\\
k_4 &= a(v_n + h\,k_3,\ p_n + h\,\ell_3), & \ell_4 &= v_n + h\,k_3,\\
v_{n+1} &= v_n + \tfrac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4), & &\\
p_{n+1} &= p_n + \tfrac{h}{6}(\ell_1 + 2\ell_2 + 2\ell_3 + \ell_4). & &
\end{aligned}$$

where $a(\cdot,\cdot)$ denotes the acceleration function (Eqs. B.12–B.11). RK4 is fourth-order accurate: the local truncation error scales as $\mathcal{O}(h^5)$.

The implementation supports multiple *sub-steps* per control time step $\Delta t$: the step size is $h = \Delta t / N_{\text{sub}}$, and $N_{\text{sub}}$ RK4 steps are applied per environment step. This allows the control step $\Delta t = 1\,\mathrm{s}$ to remain physically meaningful while maintaining numerical stability.

## B.10 Physical Parameters Summary

Table B.2 summarizes all physical parameters together with their values, units, and sources.

## B.11 Summary of Modeling Approximations

The following simplifying assumptions were made deliberately to keep the model tractable for RL training. Each is listed alongside its expected impact on physical fidelity.

**Table B.2** – Physical parameters used in the simulation.

| Symbol | Value | Unit | Source | Comment |
|---|---|---|---|---|
| $m$ | 73 500 | kg | (Airbus S.A.S., 2018) | A320-200 MTOW $\approx$ 77 000 kg; value represents a typical take-off mass |
| $T_{\text{SL}}$ | 240 000 | N | (CFM International, 2010) | 2 $\times$ CFM56-5B, 120 kN each |
| $g$ | 9.81 | m s$^{-2}$ | ISA | Standard gravity |
| $S_w$ | 122.6 | m$^2$ | (Airbus S.A.S., 2018) | A320 wing reference area |
| $S_s$ | 27 | m$^2$ | (Torenbeek, 1982) | Estimated horizontal tail area |
| $S_e$ | 10 | m$^2$ | (Torenbeek, 1982) | Estimated elevator area |
| $C_{D,0}$ | 0.02 | – | (Raymer, 2018) | Zero-lift drag, clean configuration |
| $k$ | 0.045 | – | (Raymer, 2018) | Induced drag factor |
| $C_{L,0}$ | 0.2 | – | (Airbus S.A.S., 2017) | Lift at zero AoA |
| $C_{L,\alpha}$ | 0.04 | deg$^{-1}$ | (Torenbeek, 1982) | Conservative lift-curve slope |
| $C_{L,\max}$ | 1.5 | – | (Airbus S.A.S., 2017) | Max lift, clean configuration |
| $\alpha_{\text{stall}}$ | 15 | deg | (Airbus S.A.S., 2017) | Clean-config. stall angle |
| $M_{\text{crit}}$ | 0.80 | – | (Airbus S.A.S., 2018) | Critical Mach number |
| $k_{\text{drag}}$ | 5.0 | – | Tuned | Wave-drag rise coefficient |
| $I_y$ | $9 \times 10^6$ | kg m$^2$ | (Roskam, 2001) | Pitch moment of inertia |
| $l_w$ | 1.5 | m | Estimated | Wing aero. center to center of mass |
| $l_t$ | 15 | m | (Torenbeek, 1982) | Tail moment arm |
| $T_0$ | 288.15 | K | (International Organization for Standardization, 1975) | ISA sea-level temperature |
| $P_0$ | 101 325 | Pa | (International Organization for Standardization, 1975) | ISA sea-level pressure |
| $\rho_0$ | 1.225 | kg m$^{-3}$ | (International Organization for Standardization, 1975) | ISA sea-level density |
| $L$ | 0.0065 | K m$^{-1}$ | (International Organization for Standardization, 1975) | ISA lapse rate |
| $R$ | 287.05 | J kg$^{-1}$ K$^{-1}$ | Standard | Specific gas constant of air |
| $\gamma_{\text{air}}$ | 1.4 | – | Standard | Heat capacity ratio of air |
| $\Delta t$ | 1 | s | Design choice | Control time step |

1. **2-D longitudinal model.** Only the symmetric (pitch–heave–surge) dynamics are simulated; roll, yaw, and sideslip are absent. This is appropriate for the altitude-tracking task studied in this thesis, where lateral motion is irrelevant.
2. **Constant mass.** Fuel burn is tracked as a state variable, but the feedback into the equations of motion is currently disabled (i.e., $\dot{m} = 0$ in Eq. B.14). Over a 1000 s episode at realistic fuel flow rates, mass change is $\lesssim 3\%$, so the error is small.
3. **Parabolic drag polar.** Equation (B.5) assumes a parabolic (quadratic in $C_L$) relationship. This is accurate at low-to-moderate lift coefficients but does not capture the non-parabolic behavior near stall or with high-lift devices deployed.
4. **Prandtl–Glauert correction only.** The Prandtl–Glauert rule (B.6) is valid in the subsonic range $M \lesssim 0.8$. Transonic corrections (e.g., Kármán–Tsien, Laitone) are not applied; instead, the quadratic wave-drag penalty (B.7) provides a phenomenological correction above $M_{\text{crit}}$.
5. **No ground effect, high-lift devices, or spoilers.** These effects are not modeled. The environment is intended for cruise and climb/descent tasks at altitude.
6. **First-order actuator lags.** Real engine and control-surface dynamics are higher-order; the first-order lag is a standard engineering approximation sufficient to introduce input latency, which is the dominant effect for RL.
7. **ISA atmosphere, no wind.** No turbulence, wind shear, or gusts are modeled. Only the standard density and temperature profiles of the ISA are used.

## B.12 Observation, Action, and Reward

### B.12.1 Observation Space

At each time step, the agent observes a 9-dimensional vector:

$$o_t = (\dot{x},\ z,\ \dot{z},\ \theta,\ \dot{\theta},\ \gamma,\ z_{\text{target}},\ \delta_p,\ \delta_s),$$

where $z_{\text{target}}$ is the target altitude. The horizontal position $x$ is excluded because it does not affect the flight dynamics or reward; the angle of attack $\alpha$ is omitted because it is algebraically redundant given $\theta$ and $\gamma$ (Eq. B.2).

### B.12.2 Action Space

At each time step, the agent outputs two scalars:

$$u_t = (\delta_p^{\text{cmd}},\ \delta_s^{\text{cmd}}) \in [0,1] \times [-1,1],$$

the *commanded* throttle and elevator angle, respectively. The actual actuator states evolve through the first-order lags of Section 8.

### B.12.3 Reward Function

A dense, shaped reward encourages altitude tracking while strongly penalizing boundary violations:

$$r_t = \begin{cases} -N_{\max} & \text{if } z_t \le z_{\min} \text{ or } z_t \ge z_{\max}, \\ \left( \dfrac{\Delta z_{\max} - |z_t - z_{\text{target}}|}{\Delta z_{\max}} \right)^{10} & \text{otherwise,} \end{cases}$$

where $\Delta z_{\max} = z_{\max} - z_{\min}$ normalizes the altitude error to $[0,1]$. The exponent of 10 strongly penalizes large deviations while providing a sharp gradient near the target altitude, shaping the learning landscape.

# Appendix C

# Complements on Chapter 8

## C.1 Proofs

### C.1.1 theorem 8.1

**Theorem 8.1 (restated).** *If $\mathcal{M}$ is a Stable-SP-MDP, all stable-policies $\pi$ such that $\mathcal{S}_\pi \subseteq \mathcal{S}_{target}$ are gain-optimal.*

**Proof.** Since $\pi$ is stable, we have that $\exists T$ such that $\forall t > T$, $s_t \in \mathcal{S}_\pi$. Moreover, we have that $\mathcal{S}_\pi \subseteq \mathcal{S}_{target}$, therefore $\exists T$ such that $\forall t > T$, $s_t \in \mathcal{S}_target$.

$$
\begin{aligned}
&\iff \exists T \text{ such that } \forall t > T, r_t^\pi(s_t) = r^* \\
&\iff \lim_{N\to\infty} \mathbb{E}\left[\sum_{t=0}^{N-1} r_t \pi(s)\right] = r^* \\
&\iff g^\pi(s) = r^* \forall s
\end{aligned}
$$

Since $r^* = \max(\mathcal{R})$, we have that $g^\pi(s) \geq g^{\pi'}(s) \forall \pi' \forall s$, therefore $\pi$ is gain-optimal.

□

### C.1.2 theorem 8.2

**Theorem 8.2 (restated).** *If $\mathcal{M}$ is a Stable-SP-MDP, then all bias-optimal policies in $\mathcal{M}$ are stable policies.*

**Proof.** Let $\mathcal{M}_{target}$ be a stable-SP-MDP (see definition 8.3), and let $\mathcal{S}_{target}$ be the set of target states:

$$\mathcal{S}_{target} = \{s \in \mathcal{S} \mid r(s) = r^*\}, \text{with } r^* = \max_{s' \in \mathcal{S}} r(s').$$

Let $\pi_{ns}$ be a non-stable policy. Let us assume that $\pi_{ns}$ is bias-optimal, we have:

$$b^{\pi_{ns}}(s) \geq b^{\pi}(s), \forall \pi \forall s$$

$$b^{\pi_{ns}}(s) - b^{\pi}(s) \geq 0, \forall \pi \forall s$$

Let $\pi^*$ be another bias-optimal policy, this time stable, $\pi^* \in \mathcal{S}_{stable}$. Since $\pi_{ns}$ and $\pi^*$ are bias-optimal, they are gain-optimal, therefore

$$g^{\pi_{ns}}(s) = g^{\pi^*}(s) = g^*(s) \forall s$$

. We get

$$b^{\pi_{ns}}(s) - b^{\pi^*}(s) = \lim_{N \to \infty} \mathbb{E}\left[\sum_{t=0}^{N-1} (r_t^{\pi_{ns}}(s) - g^*(s) - (r_t^{\pi^*}(s) - g^*(s)))\right]$$

$$b^{\pi_{ns}}(s) - b^{\pi^*}(s) = \lim_{N \to \infty} \mathbb{E}\left[\sum_{t=0}^{N-1} (r_t^{\pi_{ns}}(s) - r_t^{\pi^*}(s))\right]$$

Since $\pi_{ns}$ is not stable, we have that

$$\lim_{N \to \infty} d(s_{t+1,i_{target}}, s_{t,i_{target}}) > 0$$

Meanwhile, since $\pi^*$ is stable, we have

$$\lim_{N \to \infty} d(s_{t+1,i_{target}}, s_{t,i_{target}}) = 0$$

From the monotonicity of the reward w.r.t the $i_{target}$-th dimension of $s$ (see definition 8.1), we have

$$\lim_{N \to \infty} r_t^{\pi_{ns}} s - r^* < 0$$

and

$$\lim_{N \to \infty} r_t^{\pi^*} s - r^* = 0$$

While $\pi^*$ is stable in the target-states, $\pi_{ns}$ will never converge to the target, and collect suboptimal rewards. Therefore:

$$b^{\pi_{ns}}(s) - b^{\pi^*}(s) = \lim_{N\to\infty} \mathbb{E}\left[\sum_{t=0}^{N-1} (r_t^{\pi_{ns}}(s) - r_t^{\pi_*}(s))\right] < 0$$

This contradicts the definition of bias-optimality for $\pi_{ns}$. Therefore $\pi_{ns}$, a non-stable policy, cannot be bias-optimal. Hence, since a policy is either stable or non-stable, and there is at least one bias-optimal policy, in a SP-MDP every bias-optimal policy is a stable-policy.

□

## C.2 Stable Policy Identification in Stable-SP-MDPs

In a constant-action-stable-SP-MDP (definition 8.3), for every target state there exists a constant policy that both reaches and maintains this state. We propose a practical procedure to identify such constant $\varepsilon$-stable policies. The key idea is to evaluate the long-term outcome of constant actions, and interpolate between them to cover the target set.

**Constant-policy rollouts.** We discretize the action space into $K$ evenly spaced actions, and for each action we simulate a trajectory, long enough for stabilization, under the corresponding constant policy. The value for the target dimension of the final state of each rollout is recorded as the asymptotic state associated with that constant action.

**Algorithm 6.** Constant-Policy Rollouts for Stabilization

**Require:** Action space $\mathcal{A}$, initial state $s_0$, number of rollouts $K$, horizon $T$

Sample evenly spaced actions $A = \{a_1, \ldots, a_K\}$ from $\mathcal{A}$
**for all** $a \in A$ **do**
  $s \leftarrow s_0$
  **for** $t = 1$ to $T$ **do**
    Apply constant action $a$ : $s \leftarrow P(s, a)$
  **end for**
  Record final state $s_T$ for action $a$
**end for**

**Interpolation under Lipschitz dynamics.** If the transition function is $L$-Lipschitz in the state variable, then we assume the constant policies corresponding to nearby asymptotic state can be interpolated. Concretely, for any state $\tilde{s}$ between equilibria $s_T(a_i)$ and $s_T(a_j)$, there exists an interpolated constant action $\tilde{a}$ that maintains $\tilde{s}$ with at most $O(L\|s_T(a_i) - s_T(a_j)\|)$ deviation. Thus, by fitting an interpolator from observed asymptotic states $\{s_T(a)\}$ to their corresponding

constant actions $\{a\}$, we obtain an approximate mapping that predicts the stabilizing constant action for any state in $\mathcal{S}_{target}$. This construction provides us with an $\varepsilon$-stable expert policy $\pi_{\text{stable}}$.

## C.3 Baselines pseudocode

**Algorithm 7.** Soft Actor-Critic (SAC) Haarnoja et al., 2018

**Initialize** policy $\pi_\theta$, critics $Q_{\phi_1}$, $Q_{\phi_2}$, targets $\bar{Q}_{\phi_1} \leftarrow Q_{\phi_1}$, $\bar{Q}_{\phi_2} \leftarrow Q_{\phi_2}$, buffer $\mathcal{D}$, temperature $\alpha$
**for** $t = 1, \dots, T$ **do**
  **— Action selection —**
  Sample action $a_t \sim \pi_\theta(\cdot \mid s_t)$
  Execute $a_t$, observe $r_t$, $s_{t+1}$
  Store $(s_t,\ a_t,\ r_t,\ s_{t+1})$ in $\mathcal{D}$
  **— Critic update —**
  Sample batch $\{(s, a, r, s')\} \sim \mathcal{D}$
  $a' \sim \pi_\theta(\cdot \mid s')$
  $y = r + \gamma\Big[\min_{i=1,2} \bar{Q}_{\phi_i}(s', a') - \alpha \log \pi_\theta(a' \mid s')\Big]$
  $\phi_i \leftarrow \phi_i - \nabla_{\phi_i}\big(Q_{\phi_i}(s, a) - y\big)^2 \quad$ for $i = 1, 2$
  **— Actor update —**
  $\tilde{a} \sim \pi_\theta(\cdot \mid s)$
  $\theta \leftarrow \theta + \nabla_\theta[\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}) - \alpha \log \pi_\theta(\tilde{a} \mid s)]$
  **— Temperature update —**
  $\alpha \leftarrow \alpha - \nabla_\alpha\Big[\alpha\Big(-\log \pi_\theta(\tilde{a} \mid s) - \bar{\mathcal{H}}\Big)\Big]$
  **— Target network update —**
  $\bar{Q}_{\phi_i} \leftarrow \tau\, \bar{Q}_{\phi_i} + (1 - \tau)\, Q_{\phi_i} \quad$ for $i = 1, 2$
**end for**

**Algorithm 8.** IBRL-style SAC (actor proposal, no bootstrap proposal)

**Initialize** policy $\pi_\theta$, critics $Q_{\phi_1}$, $Q_{\phi_2}$, targets $\bar{Q}_{\phi_1} \leftarrow Q_{\phi_1}$, $\bar{Q}_{\phi_2} \leftarrow Q_{\phi_2}$, buffer $\mathcal{D}$, temperature $\alpha$
**Require** expert policy $\pi^e$
**for** $t = 1, \dots, T$ **do**
  **— Action selection —**
  $a_t^{\text{exp}} \leftarrow \pi^e(s_t)$
  $a_t^{\pi} \sim \pi_\theta(\cdot \mid s_t)$
  $a_t \leftarrow \arg\max_{a \in \{a_t^{\text{exp}},\, a_t^{\pi}\}} \min_{i=1,2} Q_{\phi_i}(s_t, a)$
  Execute $a_t$, observe $r_t$, $s_{t+1}$
  Store $(s_t,\ a_t,\ r_t,\ s_{t+1})$ in $\mathcal{D}$ ▷ executed action stored (expert or policy)
  **— Critic update —**
  Sample batch $\{(s, a, r, s')\} \sim \mathcal{D}$
  $a' \sim \pi_\theta(\cdot \mid s')$
  $y = r + \gamma \Big[\min_{i=1,2} \bar{Q}_{\phi_i}(s', a') - \alpha \log \pi_\theta(a' \mid s')\Big]$
  $\phi_i \leftarrow \phi_i - \nabla_{\phi_i} \big(Q_{\phi_i}(s, a) - y\big)^2$ for $i = 1, 2$
  **— Actor update —**
  $\tilde{a} \sim \pi_\theta(\cdot \mid s)$
  $\theta \leftarrow \theta + \nabla_\theta [\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}) - \alpha \log \pi_\theta(\tilde{a} \mid s)]$
  **— Temperature update —**
  $\alpha \leftarrow \alpha - \nabla_\alpha \Big[\alpha \Big(-\log \pi_\theta(\tilde{a} \mid s) - \bar{\mathcal{H}}\Big)\Big]$
  **— Target network update —**
  $\bar{Q}_{\phi_i} \leftarrow \tau\, \bar{Q}_{\phi_i} + (1 - \tau)\, Q_{\phi_i}$ for $i = 1, 2$
**end for**

**Algorithm 9.** Residual SAC  Johannink et al., 2019

**Initialize** policy $\pi_\theta$, critics $Q_{\phi_1}$, $Q_{\phi_2}$, targets $\bar{Q}_{\phi_1} \leftarrow Q_{\phi_1}$, $\bar{Q}_{\phi_2} \leftarrow Q_{\phi_2}$, buffer $\mathcal{D}$, temperature $\alpha$
**Require** expert policy $\pi^e$
**for** $t = 1, \dots, T$ **do**
  **— Action selection —**
  $a_t^{\text{res}} \sim \pi_\theta(\cdot \mid s_t)$
  $a_t \leftarrow \text{clip}(\pi^e(s_t) + a_t^{\text{res}},\ -1,\ 1)$
  Execute $a_t$, observe $r_t$, $s_{t+1}$
  Store $(s_t,\ a_t,\ r_t,\ s_{t+1})$ in $\mathcal{D}$  ▷ full executed action stored
  **— Critic update —**
  Sample batch $\{(s, a, r, s')\} \sim \mathcal{D}$
  $a^{\text{res}'} \sim \pi_\theta(\cdot \mid s')$
  $a' \leftarrow \text{clip}\Big(\pi^e(s') + a^{\text{res}'},\ -1,\ 1\Big)$
  $y = r + \gamma\Big[\min_{i=1,2} \bar{Q}_{\phi_i}(s', a') - \alpha \log \pi_\theta(a^{\text{res}'} \mid s')\Big]$
  $\phi_i \leftarrow \phi_i - \nabla_{\phi_i}\big(Q_{\phi_i}(s, a) - y\big)^2 \quad$ for $i = 1, 2$
  **— Actor update —**
  $a^{\text{res}} \sim \pi_\theta(\cdot \mid s)$
  $\tilde{a} \leftarrow \text{clip}(\pi^e(s) + a^{\text{res}},\ -1,\ 1)$
  $\theta \leftarrow \theta + \nabla_\theta[\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}) - \alpha \log \pi_\theta(a^{\text{res}} \mid s)]$
  **— Temperature update —**
  $\alpha \leftarrow \alpha - \nabla_\alpha\Big[\alpha\Big(-\log \pi_\theta(a^{\text{res}} \mid s) - \bar{\mathcal{H}}\Big)\Big]$
  **— Target network update —**
  $\bar{Q}_{\phi_i} \leftarrow \tau\, \bar{Q}_{\phi_i} + (1 - \tau)\, Q_{\phi_i} \quad$ for $i = 1, 2$
**end for**

# Appendix: Hyperparameter Search Methodology

## Algorithm and Environment

We tune vanilla SAC (no expert guidance) on the `Plane` environment with target and initial altitudes drawn uniformly from $[3{,}000, 8{,}000]$ m and episodes capped at $10{,}000$ steps. The optimization objective is the mean episodic reward evaluated over 25 test episodes, logged every $10{,}000$ environment steps.

## Search Algorithm

We use **Optuna's (Akiba et al., 2019) Tree-structured Parzen Estimator (TPE)** (Bergstra et al., 2011). After an initial random phase of roughly 10 trials, TPE fits two kernel density estimates

$p(\mathbf{x} \mid \text{good})$ and $p(\mathbf{x} \mid \text{bad})$ from completed trials and proposes the next configuration by maximizing the ratio $\frac{p(\mathbf{x}|\text{good})}{p(\mathbf{x}|\text{bad})}$.

### Two-Phase Protocol

Because our SAC training loop is a monolithic JAX-JIT call without mid-training callbacks, the standard asynchronous successive halving (ASHA) method cannot prune running trials. We instead adopt a two-phase design:

**Phase 1 — Exploration:** Each of the $N = 50$ TPE trials is trained for $3 \times 10^5$ **timesteps** over **5 independent seeds**. The trial score is the final (last-logged) value of the mean episodic reward, averaged across seeds.

**Phase 2 — Confirmation:** The top-$K$ configurations from Phase 1 are re-trained for $10^6$ **timesteps** over **15 independent seeds**. The trial score is the mean episodic reward averaged over all evaluation points in the last $2 \times 10^5$ timesteps and across all seeds, matching the evaluation window used in the final experiments.

### Search Space

Table C.1 lists the full search space. Continuous ranges marked *log* are sampled uniformly on a log scale.

All actor and critic networks are two-hidden-layer Multi-Layer Perceptron (MLP)s with ReLU activations and equal width $w \in \{256, 512\}$, i.e. architecture $(w, \texttt{relu}, w, \texttt{relu})$. Gradient clipping, when enabled, clips by global $\ell_2$ norm.

### Compute Budget

Phase 1 consumes at most $40 \times 5 \times 3 \times 10^5 = 6 \times 10^7$ environment steps. Phase 2 (top-$K = 5$) adds at most $5 \times 15 \times 10^6 = 7.5 \times 10^7$ steps.

## Appendix: Hyperparameter Tables

All experiments share the same environment: `Plane` with target and initial altitudes in $[3{,}000, 8{,}000]$ m, episodes of at most $10{,}000$ steps, and $1 \times 10^6$ total training timesteps. Reward is evaluated over 25 test episodes after every $10{,}000$ environment steps.

### SAC Hyperparameters

Table C.2 lists the hyperparameters selected by the Bayesian search described above; Table C.3 lists the remaining hyperparameters that were held fixed across all experiments.

**Table C.1** – Hyperparameter search space for SAC on `Plane`.

| Hyperparameter | Type | Range / Values | Scale |
|---|---|---|---|
| Actor learning rate | Continuous | $[10^{-4}, 10^{-3}]$ | log |
| Critic learning rate | Continuous | $[10^{-4}, 10^{-3}]$ | log |
| Entropy coefficient lr | Continuous | $[10^{-4}, 10^{-3}]$ | log |
| Discount factor $\gamma$ | Continuous | $[0.97, 0.999]$ | linear |
| Polyak coefficient $\tau$ | Continuous | $[5 \times 10^{-4}, 5 \times 10^{-2}]$ | log |
| Initial entropy coeff $\alpha_0$ | Continuous | $[0.05, 2.0]$ | log |
| Target entropy per dim | Continuous | $[-4.0, -0.2]$ | linear |
| Batch size | Categorical | $\{256, 512\}$ | — |
| Network width | Categorical | $\{256, 512\}$ | — |
| Gradient clipping | Categorical | $\{\text{none}, 0.5, 1.0\}$ | — |

### EDGE Hyperparameters

Table C.4 reports the hyperparameters for **EDGE** (`EDGE _decay_050`), the main result. EDGE gates between the expert action and the learned policy action using an $\varepsilon$-greedy scheme: with probability $\varepsilon_t$ the expert action $\pi^e(s_t)$ is selected, and with probability $1-\varepsilon_t$ an action is sampled from the learned policy $\pi_\theta$. The mixing probability $\varepsilon_t$ decays linearly from $\varepsilon_0$ to $0$ over the first $f_{\text{decay}}$ fraction of training.

### IBRL and Residual Policy Hyperparameters

Table C.5 lists the hyperparameters specific to the two additional baselines. IBRL Hu, Mirchandani, and Sadigh, 2023 gates deterministically: the expert action is taken whenever $Q(s, \pi^e) > Q(s, \pi_\theta)$ (argmax, no decay). Residual RL (Johannink et al., 2019) outputs a residual $a_\delta$ that is added to the (frozen) expert action before environment execution: $a = \text{clip}(a_{\pi^e} + a_\delta, -1, 1)$.

**Table C.2** – Tuned SAC hyperparameters (Phase 2 best configuration).

| Hyperparameter | Value | Notes |
|---|---|---|
| Actor learning rate | $3.41 \times 10^{-4}$ | log-uniform search |
| Critic learning rate | $2.55 \times 10^{-4}$ | log-uniform search |
| Entropy coeff. lr | $6.13 \times 10^{-4}$ | log-uniform search |
| Discount $\gamma$ | $0.9805$ | linear search |
| Polyak $\tau$ | $8.36 \times 10^{-3}$ | log-uniform search |
| Initial ent. coeff. $\alpha_0$ | $1.207$ | log-uniform search |
| Target entropy per dim | $-0.268$ | linear search |
| Batch size | $256$ | categorical |
| Network width | $256$ | categorical |
| Gradient clip ($\ell_2$) | $1.0$ | categorical |

## C.4 Effect of TD-target bootstrap on IBRL and EDGE

In Section 8.5.1, we implement the IBRL baseline without the TD-target bootstrap (i.e., without selecting the next-state action as the argmax over expert and policy in critic updates), to isolate the comparison to the gating mechanism alone. Symmetrically, our EDGE implementation does not include a bootstrap variant either. To verify that this simplification does not bias the conclusions, we compare all four combinations: IBRL and EDGE each with and without TD-target bootstrap, using the same experimental protocol (100 seeds, $10^6$ steps, same hyperparameters). In the bootstrap variants, the next-state action in the critic update is selected using each method's own gating mechanism (argmax for IBRL, stochastic $\varepsilon$-greedy for EDGE).

**Table C.3** – Fixed SAC hyperparameters (not tuned, shared by all algorithms).

| Hyperparameter | Value | Notes |
|---|---|---|
| Replay buffer size | 1,000,000 | transitions |
| Learning starts | 10,000 | steps before first update |
| Number of critics | 2 | double Q-learning |
| Network architecture | $(w, \texttt{ReLU}, w, \texttt{ReLU})$ | $w = 256$ |
| Evaluation episodes | 25 | per checkpoint |
| Log / eval frequency | 10,000 steps | |
| Training timesteps | $1 \times 10^6$ | |

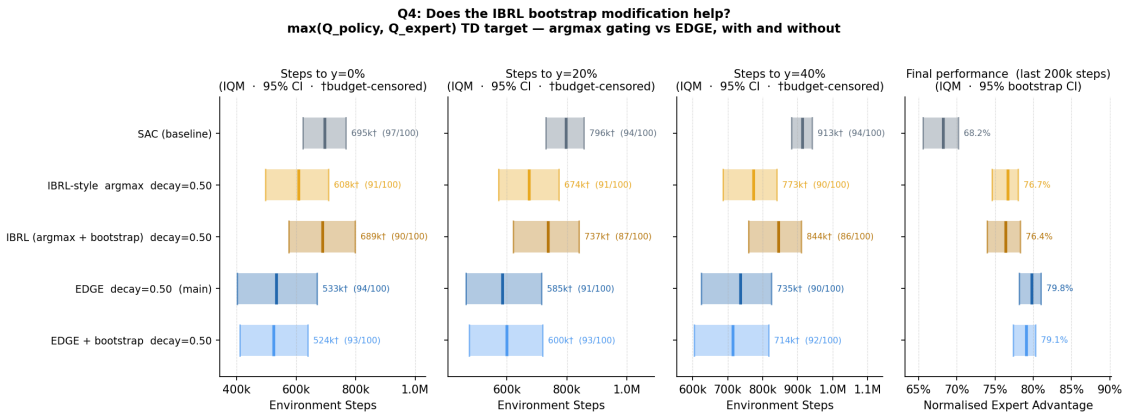



**Figure C.1** – Q4: Effect of TD-target bootstrap on IBRL and EDGE. Each bar shows the IQM (vertical line) and 95% bootstrap CI (shaded rectangle) across 100 seeds. The first three panels report steps to reach NEA $\geq 0\%$, 20%, and 40% (lower is better); the rightmost panel reports final NEA IQM over the last 200k steps (higher is better). No systematic difference is observed between the bootstrap and no-bootstrap variants of either method, confirming that omitting the TD-target bootstrap does not bias the comparison.

For both IBRL and EDGE, the bootstrap and no-bootstrap variants are indistinguishable across all metrics, with confidence intervals overlapping substantially and no consistent ordering visible. This empirically confirms that the TD-target bootstrap does not materially affect performance in this setting for either gating mechanism, validating our decision to omit it for simplicity and to keep the comparison focused on the gating mechanism itself.

**Table C.4** – EDGE hyperparameters (main result: `EDGE _decay_050`).

| Hyperparameter | Value | Notes |
|---|---|---|
| Gating mode | $\varepsilon$-greedy | linear decay of expert probability |
| Initial expert probability $\varepsilon_0$ | 0.5 | |
| Decay fraction $f_{\text{decay}}$ | 0.50 | fraction of total training |
| Critic updates per step | 1 | |
| Policy update start | 2,000 steps | |
| Alpha update start | 2,000 steps | |

**Table C.5** – Hyperparameters for IBRL and Residual RL baselines.

| Hyperparameter | IBRL | Residual RL |
|---|---|---|
| Gating / action mode | Argmax (deterministic) | $\text{clip}(a_{\pi^e} + a_\delta, -1, 1)$ |
| Critic updates per step | 1 | |
| Policy update start | 2,000 steps | 10,000 steps |
| Alpha update start | 2,000 steps | 10,000 steps |

## C.5 Statistical test methodology

### Overview

All pairwise comparisons are made between EDGE (`EDGE _decay_050`, the main result) and each of the three baselines: vanilla SAC, Residual RL, and IBRL. Using a single reference method eliminates logical constraints between the tested hypotheses, so we apply the Holm–Bonferroni (Lehmann and Romano, 2022) step-down procedure for multiplicity control. Each family therefore contains $K = 3$ comparisons and controls the family-wise error rate at $\alpha = 0.05$.

### Statistical test: IQM

The IQM trims the bottom and top $25\%$ of the seed distribution and averages the remainder. For a sorted sample $x_{(1)} \leq \cdots \leq x_{(n)}$:

$$\mathrm{IQM}(x) = \frac{1}{h-l} \sum_{i=l}^{h-1} x_{(i)}, \quad l = \lfloor 0.25\, n \rfloor,\ h = \lceil 0.75\, n \rceil.$$

We operate on per-seed scalar summaries (described below) so that each seed contributes exactly one observation to the IQM. With $100$ seeds the IQM discards the $25$ lowest- and $25$ highest-performing seeds, averaging the central $50$.

### Test Families

Two separate families of tests are conducted on the **Normalized Expert Advantage** metric, defined as

$$\mathrm{NEA}_t = \frac{\bar{r}_t^{\pi} - \bar{r}_t^{*}}{\mathrm{G}^{\max} - \bar{r}_t^{*}},$$

where $\bar{r}_t^{\pi}$ is the policy's mean episodic reward at step $t$, $\bar{r}_t^{*}$ is the expert's mean episodic reward, and $G^{max}$ is the theoretical maximum. Higher values indicate a policy closer to expert-level performance; $\mathrm{NEA} = 0$ means the policy matches the expert.

**Asymptote.** For each seed, the IQM of all evaluation checkpoints in the last $200{,}000$ environment steps is computed, yielding one scalar per seed. This measures the long-run performance of the policy.

**Steps to threshold.** For each seed, the first step at which the metric permanently crosses a threshold $y^*$ is recorded. Seeds that never cross $y^*$ within the $10^6$-step budget are **budget-censored**: they receive the maximum value $10^6$ (equivalent to RMST-style imputation). Thresholds $y^* \in \{0.0,\ 0.2,\ 0.4\}$ are tested, measuring how quickly each method reduces expert bias to near zero, to $20\%$, and to $40\%$ of the expert score.

### Permutation Test

For a comparison between groups $A$ (EDGE) and $B$ (baseline) with per-seed observations $\mathbf{x}_A$ and $\mathbf{x}_B$, the observed test statistic is

$$T_{\mathrm{obs}} = \mathrm{IQM}(\mathbf{x}_A) - \mathrm{IQM}(\mathbf{x}_B).$$

Under the null hypothesis $H_0$: the two groups are exchangeable, we pool $\mathbf{x} = [\mathbf{x}_A; \mathbf{x}_B]$ and draw $B = 10{,}000$ permutations. For each permutation $\pi^{(b)}$ the statistic $T^{(b)} = \mathrm{IQM}(\pi^{(b)}_{1:|A|}) -$

$\mathrm{IQM}(\pi^{(b)}_{|A|+1:})$ is computed. The two-sided Monte Carlo p-value is

$$p_{\text{raw}} = \frac{1}{B}\sum_{b=1}^{B} \mathbf{1}\Big[|T^{(b)}| \geq |T_{\text{obs}}|\Big] .$$

The random number generator is seeded at 42 for reproducibility.

### Holm–Bonferroni Correction

Let $p_{(1)} \leq p_{(2)} \leq p_{(3)}$ be the sorted raw p-values for the three comparisons and let $H_{(k)}$ denote the corresponding hypothesis. The Holm–Bonferroni adjusted p-values are

$$\tilde{p}_{(k)} = \max_{j \leq k} \min\Big(1,\ p_{(j)} \cdot (K - j + 1)\Big) , \quad k = 1, 2, 3.$$

$H_{(k)}$ is rejected if $\tilde{p}_{(k)} < 0.05$. This step-down procedure controls the Family-Wise Error Rate (FWER) at $\alpha = 0.05$ under arbitrary dependence between tests.

### Results

Tables C.6 and C.7 report the outcomes. $\Delta\,\mathrm{IQM} = \mathrm{IQM}_{\mathrm{EDGE}} - \mathrm{IQM}_{\mathrm{baseline}}$; positive values indicate EDGE outperforms the baseline.

| **Group 1** | **Group 2** | $\Delta$ IQM | $p^{\text{adj}}$ | Significant |
|---|---|---|---|---|
| EGE decay=0.50 (main) | SAC (baseline) | +11.6% | 0.0000 | **Yes** |
| EGE decay=0.50 (main) | Residual RL | +4.0% | 0.0002 | **Yes** |
| EGE decay=0.50 (main) | IBRL-style argmax (no decay) | +2.9% | 0.0091 | **Yes** |

**Table C.6** – IQM permutation test + Holm-Bonferroni for asymptotic performance (IQM over last 200k steps, $\alpha = 0.05$, 10,000 permutations). $p$-values adjusted for multiple comparisons; significant differences ($p^{\text{adj}} < 0.05$) in bold.

**NEA $\geq 0\%$**

| Group 1 | Group 2 | $\Delta$ IQM | $p^{\text{adj}}$ | Significant |
|---|---|---|---|---|
| EGE decay=0.50 (main) | SAC (baseline) | -161,300 steps | 0.1068 | No |
| EGE decay=0.50 (main) | Residual RL | +27,000 steps | 0.7508 | No |
| EGE decay=0.50 (main) | IBRL-style argmax (no decay) | -148,400 steps | 0.2018 | No |

**NEA $\geq 20\%$**

| Group 1 | Group 2 | $\Delta$ IQM | $p^{\text{adj}}$ | Significant |
|---|---|---|---|---|
| EGE decay=0.50 (main) | SAC (baseline) | -211,300 steps | 0.0039 | **Yes** |
| EGE decay=0.50 (main) | Residual RL | -94,300 steps | 0.2313 | No |
| EGE decay=0.50 (main) | IBRL-style argmax (no decay) | -139,600 steps | 0.1878 | No |

**NEA $\geq 40\%$**

| Group 1 | Group 2 | $\Delta$ IQM | $p^{\text{adj}}$ | Significant |
|---|---|---|---|---|
| EGE decay=0.50 (main) | SAC (baseline) | -177,600 steps | 0.0000 | **Yes** |
| EGE decay=0.50 (main) | Residual RL | -107,600 steps | 0.0498 | **Yes** |
| EGE decay=0.50 (main) | IBRL-style argmax (no decay) | -69,000 steps | 0.2823 | No |

**Table C.7** – IQM permutation test + Holm-Bonferroni for steps to first reach each NEA threshold (budget-censored, $\alpha = 0.05$, 10,000 permutations). $p$-values adjusted for multiple comparisons; significant differences ($p^{\text{adj}} < 0.05$) in bold.

## C.6 Hardware and Computations

### C.6.1 Hardware

The experiments were run on $4\times$ NVIDIA L40S GPUs on the Scool team server "bob", while almost all computation time is done on GPU, the compilation was still performed on a AMD EPYC 9454 48-Core Processor.

### C.6.2 Computation time

**Table C.8** – Cumulative compute budget. Wall-clock GPU-hours are reported; for the main/ablation/degradation experiments, seed-hours are divided by 100 since seeds were evaluated in batches of 100 in parallel, except for the hyperparameter search (5 seeds in parallel for Phase 1, 15 for Phase 2).

| Category | Configurations | Seeds | GPU-hours |
|---|---|---|---|
| Hyperparameter search (Phase 1) | 50 trials | 250 | 9.4 |
| Hyperparameter search (Phase 2) | 10 trials | 150 | 9.9 |
| Main results | 4 | 400 | 10.6 |
| Ablation study | 23 | 2,400 | 55.4 |
| Degradation study | 18 | 1,800 | 51.0 |
| **Total** | **105** | **5,000** | **136.3** |

### C.6.3 Energy consumption

**Table C.9** – Cumulative compute budget, estimated energy consumption, and $CO_2$ emissions. All experiments ran on a single server (`scool-bob`) equipped with 4× NVIDIA L40S GPUs (350 W TDP each) and 2× AMD EPYC 9454 CPUs (290 W TDP each). Wall-clock GPU-hours for the main/ablation/degradation experiments are total seed-hours divided by 100 (seeds evaluated in batches of 100 in parallel). Hyperparameter search trials were run sequentially with 5 (Phase 1) or 15 (Phase 2) seeds in parallel per trial. Energy is estimated as wall-clock time × effective system power, assuming 80 % GPU utilisation, 70 % CPU utilisation for batch runs (30–55 % for the search), a system overhead of 150 W, and a PUE of 1.3. $CO_2$ figures use the 2024 French grid intensity of 85 g $CO_2$eq kWh$^{-1}$ (nuclear-heavy); EU and global averages are provided for reference.

| **Category** | **Config.** | **Seeds** | **GPU-h** | **Energy (kWh)** |
|---|---|---|---|---|
| HP search — Phase 1 (5 seeds/trial) | 50 trials | 250 | 9.4 | 9 |
| HP search — Phase 2 (15 seeds/trial) | 10 trials | 150 | 9.9 | 19 |
| Main results | 4 | 400 | 10.6 | |
| Ablation study | 23 | 2,400 | 55.4 | 255 |
| Degradation study | 18 | 1,800 | 51.0 | |
| **Total** | **105** | **5,000** | **136.3** | **283** |
| $CO_2$ — France (85 g/kWh) | | | | 24 kg |
| $CO_2$ — EU avg. (233 g/kWh) | | | | 66 kg |
| $CO_2$ — Global (460 g/kWh) | | | | 130 kg |

# Appendix D

# Use of Generative AI Tools

I chose to include this appendix on my own initiative, not because my institution requires it. Throughout this PhD, I kept my supervisors fully informed of my use of Generative Artificial Intelligence (GenAI) tools, and this appendix is a natural extension of that transparency to the reader. I believe that GenAI assistance has had a significant enough impact on this work, most particularly in dramatically reducing the time between ideation and implementation, that it deserves to be disclosed honestly rather than passed over in silence. I am also aware of the risks these tools carry in terms of integrity, plagiarism, and the risk of hindering the learning process a PhD is meant to provide. I am confident that my usage did not fall into any of those traps, but I also want to be honest that some results would likely not have been obtained in the same way or within the same timeframe without GenAI, and that the manuscript itself would probably have been somewhat different. Those differences are, I believe, purely editorial: everything in this thesis reflects my own views and original work, with one exception regarding ideation that I describe below. The full breakdown of how I used GenAI follows, and I take full responsibility for every word of this thesis regardless of what assisted in producing it.

## Writing Assistance

I used GenAI language models (primarily Claude by Anthropic) as writing assistants during the drafting and revision of this manuscript, for the following tasks:

- Proofreading and grammar correction of English prose.
- Rephrasing sentences for clarity and conciseness.
- Checking consistency of notation and terminology across chapters.
- Identifying and correcting duplicate bibliography entries.
- Reviewing of the manuscript.

- Drafting some paragraphs based on my notes, primarily in this appendix and in the conclusion chapter.
- Creation and modification of LaTeX code for nicer visual formatting.

All theoretical results and their formalization are entirely my own work, with no GenAI involvement. On the question of ideation, I did use GenAI on one occasion, in the context of Chapter 8, as a discussion partner to help diagnose the failure modes of mechanisms I had previously conceived myself. By that point, the research subject, the literature review, and the theoretical formalization had all been completed without GenAI. This happened after I had genuinely exhausted my own candidate ideas. Through that conversation, a list of candidate directions emerged iteratively, one of which I ultimately selected, improved, and pursued. I do not consider this a direct solicitation of ideas: it only happened because my own prior ideas had already failed, it required numerous exchanges to build a shared understanding of those failure mechanisms, and the direction I ended up following underwent substantial modifications on my part before it yielded any results. In that sense, I think of this interaction as the kind of input a discussion with a colleague might have provided. All other ideation was done entirely without GenAI assistance.

## Literature Review

I conducted the literature review using standard academic search tools. I did occasionally ask GenAI to suggest additional references I might have missed, and in every case I independently verified and read each suggested work before including it. I should note that GenAI did sometimes hallucinate papers, so verification is genuinely necessary, especially when a citation seems suspiciously well-aligned with the topic at hand. That said, this was quite rare: two instances come to mind across the entire thesis.

The most significant case of GenAI-assisted literature discovery occurred in Chapter 8, where GenAI pointed me to Hu, Mirchandani, and Sadigh (2023) (IBRL), a paper I had not found on my own. I attribute this to the fact that the work was published outside the main machine learning venues, and that its framing as imitation learning from human demonstrations in robotics made it very hard to surface through my searches oriented toward reinforcement learning, industrial processes, or expert knowledge. Even after a thorough review, I had missed it. Without that suggestion, I might have made an erroneous novelty claim, and I am glad I did not.

## Code and Experiments

I wrote the base code and all core algorithms entirely on my own. I did use GenAI coding assistants in the context of Chapter 8 to accelerate the implementation and testing of experimental modifications: concretely, GenAI helped add the modifications required by Chapter 8 on top of the SAC baseline, and produce code used to run experiments and generate figures. No experimental data was generated by GenAI; its role was purely in the implementation layer, and every GenAI-suggested modification was reviewed and validated by me before use.

I want to be clear that GenAI assistance was genuinely instrumental here: it allowed me to implement and test ideas much faster than I could have alone (tasks that would have taken days were completed in minutes), and to prune unsuccessful modifications in significantly less time. That acceleration was a meaningful factor in the experimental throughput of Chapter 8.

## Learning Experience Without GenAI

I made a deliberate choice to avoid GenAI tools for the vast majority of this PhD, only adopting them significantly in the final month of thesis writing where efficiency became the primary constraint. I was conscious early on that delegating work to GenAI would mean not experiencing it myself, and I did not want to shortcut the learning that comes from struggling with implementation, debugging, and writing.

To be specific, the following was done entirely without GenAI assistance. The entirety of Chapter 3, including the identification of the airplane proxy and its implementation. The entirety of the EVarEst work, from implementation and experiments to manuscript writing, submitted to conferences independently and ultimately presented at EWRL 2025. The initial version of what would become EDGE. More broadly, all open-source research code was written without GenAI assistance; the only exceptions are visual aspects (PyGame rendering of TargetGym environments for agent behavior videos), refactoring tasks, and the implementation of experimental modifications on top of existing baselines, all of which occurred at the very end of the PhD and are scoped to Chapter 8.

For completeness: I did experiment with OpenAI's ChatGPT and Google's Gemini at various points, but ChatGPT was not used for any work in this thesis, and Gemini was only used sporadically for literature assistance (fetching BibTeX entries, identifying published versions of arXiv papers). The tool I relied upon for essentially all GenAI-assisted work was Anthropic's Claude (Sonnet 4.6), including Claude Code.

# List of Figures

2.1 Deep RL algorithm map organized by PG estimator (columns) and critic objective (rows). on-policy and off-policy cells encode the data regime; gray cells represent structurally incompatible or purely tabular combinations. **Boxed bold** names are algorithms central to this thesis; plain names provide structural context. ACER (off-policy likelihood-ratio with Bellman $Q$) remains niche because importance sampling variance dominates at high replay ratios. SQL (Soft Q-Learning) uses a soft Bellman objective with an implicit policy derived directly from the $Q$-function rather than a separate actor network; it has been largely superseded by SAC. . . . . 19

3.1 Top down view of the whole flat glass production line. From Saint-Gobain, 2021 28

3.2 Views from the raw material preparation. From Saint-Gobain, 2021 . . . . . 29

3.3 Side-by-side views of the furnace. From Saint-Gobain, 2021 . . . . . 30

3.4 Side-by-side views of the float bath. From Saint-Gobain, 2021 . . . . . 30

3.5 Front and side view of a top-roll. From Saint-Gobain, 2021 . . . . . 31

3.6 Side view of the lehr, measuring from 150m to 200m long. From Saint-Gobain, 2021 . . . . . 31

3.7 Views from cold end. From Saint-Gobain, 2021 . . . . . 32

3.8 Example altitude trajectories of the aircraft simulation under different power inputs, illustrating the stability around the setpoint (Berthelot, 2025b). . . . . 41

4.1 Taxonomy of sample-efficient RL algorithms organized by **how the model is used** (columns) and the **Update-To-Data (UTD) ratio** (y-axis, approximate log scale). Three mechanistically distinct model-based families are distinguished: **Synthetic Data** (the model generates fake transitions added to a replay buffer and consumed by a model-free update); **Planning** (the model is queried at decision time to search over action sequences via MCTS or CEM); and **Imagination/BPTT** (the policy is trained **through** full imagined trajectories in a learned latent space via backpropagation through time). Algorithms placed between two columns combine both mechanisms. Two key observations emerge: (i) high sample efficiency does not require a model (REDQ/DroQ at UTD=20); (ii) model-based methods largely achieve sample efficiency at UTD≈1, relying on model quality rather than update frequency. UTD values are for illustrating the ranking among algorithms rather than the exact values. . . . . 50

4.2 Example state vector for autonomous driving. Features with high RL value (amber) are those most relevant for learning a good policy, while low-value features (gray) carry little information for the agent despite being part of the observation. . . . . 53

5.1 MuJoCo locomotion tasks (left to right): HalfCheetah, Hopper, Walker2d, Ant, Humanoid. . . . . 72

5.2 AdroitHand dexterous manipulation tasks (left to right): Door, Hammer, Pen, Relocate. . . . . 73

5.3 Average performance over the last $10^5$ training timesteps for EVarEst-SAC (left) and EVarEst-PPO (right) on MuJoCo with different $\alpha_{\text{ev}}$ values. Y-axis: Average episodic returns. Black lines represent the metric specified at the top of the column over 10 seeds. Bars represent 95% bootstrap confidence intervals. Pink is the best performing $\alpha_{\text{ev}}$, dark blue are the baselines. . . . . 73

5.4 Average performance over the last $10^5$ training timesteps for EVarEst-SAC (left) and EVarEst-PPO (right) on AdroitHand with different $\alpha_{\text{ev}}$ values. Y-axis: Average episodic returns. Black lines represent the metric specified at the top of the column over 10 seeds. Bars represent 95% bootstrap confidence intervals. Pink is the best performing $\alpha_{\text{ev}}$, dark blue are the baselines. . . . . 74

5.5 Feature rank of EVarEst-SAC evolution during training (Top: Mujoco, Bottom: AdroitHand) . . . . . 76

5.6 Relative difference between EVarEst-SAC and baseline MSE value networks for value estimation metrics (MSE, Bias$^2$ and VRE) averaged on the last $10^5$ timesteps with different $\alpha_{\text{ev}}$ values . . . . . 78

5.7 Relative difference between EVarEst and baseline MSE for value loss over training for different $\alpha_{\text{ev}}$ values on Mujoco tasks (Top: SAC, Bottom: PPO) . . . . . 79

6.1 Key milestones in the history of parallelization applied to deep RL. Color indicates whether the algorithm is on-policy (teal) or off-policy (orange). The progression reflects a shift from distributed CPU infrastructures toward massively parallel single-GPU pipelines, and more recently from on-policy dominance toward off-policy methods. . . . . 90

6.2 Hybrid Sim2Real RL Architecture: The left side of this diagram depicts the massively parallel training of a fast agent. This allows for rapid training as well as building robustness w.r.t. model misspecification. The policy learned in simulation is then transferred to a more sample-efficient agent for fine-tuning on the real system. . . . . 91

6.3 Schematic representation of temporal bottlenecks in the CPU-GPU pipeline. Timings are provided for illustrative purposes. . . . . 94

6.4 Wall-clock time comparison between a pure CPU pipeline and a hybrid CPU+GPU pipeline on the same RL workload. The bottleneck induced by host-device transfers negates most of the expected GPU benefit. Measured on SB3 SAC on Gymnasium (MuJoCo) Ant with an Nvidia L40S GPU and AMD EPYC 9454 48-Core Processor CPU . . . . . 94

6.5 Schematic representation of the Jax compilation process . . . . . 97

6.6 Algorithmic properties evaluated using `bsuite`, taken from Osband et al. (2019).103

7.1 Representation of the impact of tricks on algorithm performance, adapted from Munroe (2020). . . . . 108

7.2 Diagnostic flowchart of the *ProbingEnvironments* test suite. Tests are run in sequence; a failure immediately identifies the suspect algorithmic component without requiring the practitioner to inspect the full codebase. . . . . 111

7.3 Wall-clock time and speedup of AJAX + Brax (end-to-end JAX) versus SB3 + Gymnasium (sequential) for SAC on the Ant environment, as a function of the number of seeds. The left axis (log scale) shows wall-clock time in hours; the right axis shows the speedup factor relative to SB3 sequential. Running with W&B logging incurs a modest overhead but does not change the scaling behavior. 116

7.4 Wall-clock time and speedup of AJAX + Brax (end-to-end JAX) versus SB3 + Gymnasium (sequential) for PPO on the Ant environment, as a function of the number of seeds. The left axis (log scale) shows wall-clock time in hours; the right axis shows the speedup factor relative to SB3 sequential. W&B logging was not ablated here, as the SAC comparison already demonstrated its negligible overhead. . . . . 116

7.5 Visual renderings of the four TargetGym environments. Top row, left to right: Plane (altitude tracking), Car (speed tracking with road gradient), Bike (upright balancing). Bottom: CSTR (concentration regulation over time). . . . . 122

7.6 Example trajectories from the four TargetGym environments. Top left: Plane (altitude tracking). Top right: Car (speed tracking with road gradient). Bottom left: Bike (upright balancing). Bottom right: CSTR (concentration regulation). 123

8.1 *Conceptual illustration.* The expert (blue) represents a stable policy that converges toward the target (dashed) but does so slowly; the learned policy (orange) reaches the same target faster, and both ultimately stabilize at the same level. *Left*: convergence over time. *Right*: the difference in sample efficiency is large (the policy reaches the target much sooner), while the asymptotic performance is the same for both. The expert's value lies in guiding early exploration toward the high-reward region; the policy is then free to surpass it in convergence speed. 138

8.2 Normalized Expert Advantage (NEA) over training, aggregated as IQM across 100 seeds. Shaded bands are 95% bootstrap confidence intervals (1 000 resamples). Higher is better; NEA = 0 indicates the policy matches the expert. . . . . 145

8.3 Summary statistics for the four methods. Each bar shows the IQM (vertical line) and 95% bootstrap CI (shaded rectangle) across 100 seeds. The first three panels report steps to reach NEA $\geq 0\%$, 20%, and 40%; the rightmost panel reports final NEA (IQM over the last 200 000 steps). Seeds that never reach a threshold within the $10^6$-step budget are budget-censored and marked with †. . . . . 146

8.4 Q1: Effect of the decay fraction $f_{\text{decay}}$ on EDGE performance (sweep values as in Q1 above), compared against the SAC baseline. . . . . 147

8.5 Q2: Sensitivity of EDGE to the initial expert probability $\varepsilon_0$ (sweep values as in Q2 above). Layout as in Figure 8.4. . . . . 148

8.6 Q3: Comparison of gating mechanisms (variants as in Q3 above). Layout as in Figure 8.3. . . . . 149

8.7 Q4: Effect of replacing the stable expert with a uniform random-action policy. Layout as in Figure 8.3. The random-expert variant performs at or below the SAC baseline across all metrics, confirming that EDGE's gains stem from the expert's domain knowledge rather than from the exploration schedule alone. . 150

8.8 Expert robustness study: each panel shows a performance metric as a function of expert noise level $\rho \in \{0\%, 2\%, 5\%, 10\%, 20\%, 40\%, 80\%\}$. The first three panels report steps to reach NEA $\geq 0\%$, $20\%$, and $40\%$ respectively (lower is better; budget-censored runs included at the training budget); the rightmost panel reports the final NEA IQM over the last 200k steps (higher is better). Lines show IQM across 100 seeds; shaded bands are 95% bootstrap CI. The SAC baseline (dashed) is noise-free throughout. . . . . 151

A.1 EVarEst-SAC critic predictions evolution during training for different $\alpha_{\text{ev}}$ values (Top: prediction mean, Bottom: prediction variance) . . . . . 168

A.2 EVarEst-PPO critic predictions evolution during training for different $\alpha_{\text{ev}}$ values (Top: prediction mean, Bottom: prediction variance) . . . . . 168

A.3 left without early termination, right with early termination. . . . . 169

A.4 Relative difference between $EVarEst$-PPO and PPO baseline for value estimation metrics (MSE, Bias$^2$ and VRE) averaged on the last $10^5$ timesteps with different $\alpha_{\text{ev}}$ values. . . . . 170

A.5 Feature rank of $EVarEst$-PPO evolution during training (Top: Mujoco, Bottom: AdroitHand). . . . . 170

A.6 Relative difference between $EVarEst$ and baseline MSE for value loss over training for different $\alpha_{\text{ev}}$ values on AdroitHand tasks (Top: SAC, Bottom: PPO). 171

A.7 Mean episodic return of $EVarEst$-SAC during training for different $\alpha_{\text{ev}}$ values(Top: Mujoco tasks, Bottom: AdroitHand tasks). . . . . 171

A.8 Mean episodic return of $EVarEst$-PPO during training for different $\alpha_{\text{ev}}$ values(Top: Mujoco tasks, Bottom: AdroitHand tasks). . . . . 172

A.9 Mean episodic length of $EVarEst$-SAC during training for different $\alpha_{\text{ev}}$ values(Top: Mujoco tasks, Bottom: AdroitHand tasks). . . . . 172

A.10 Mean episodic length of $EVarEst$-PPO during training for different $\alpha_{\text{ev}}$ values(Top: Mujoco tasks, Bottom: AdroitHand tasks). . . . . 173

B.1 Relationship between pitch angle $\theta$ (body axis relative to horizon), flight-path angle $\gamma$ (velocity vector relative to horizon), and angle of attack $\alpha = \theta - \gamma$ (body axis relative to velocity vector). All angles are positive in the nose-up/climb direction. . . . . 179

C.1 Q4: Effect of TD-target bootstrap on IBRL and EDGE. Each bar shows the IQM (vertical line) and 95% bootstrap CI (shaded rectangle) across 100 seeds. The first three panels report steps to reach NEA $\geq$ 0%, 20%, and 40% (lower is better); the rightmost panel reports final NEA IQM over the last 200k steps (higher is better). No systematic difference is observed between the bootstrap and no-bootstrap variants of either method, confirming that omitting the TD-target bootstrap does not bias the comparison. . . . . 200

# List of Algorithms


1 Proximal Policy Optimization (PPO) . . . . . 21
2 Soft Actor-Critic (SAC) . . . . . 22
3 Generic Deep RL Backbone . . . . . 86
4 Hybrid Wall-Clock to Sample-Efficient (W2S) Pipeline . . . . . 92
5 Expert Decayed-Guided-Exploration SAC (EDGE-SAC) . . . . . 140
6 Constant-Policy Rollouts for Stabilization . . . . . 193
7 Soft Actor-Critic (SAC) Haarnoja et al., 2018 . . . . . 194
8 IBRL-style SAC (actor proposal, no bootstrap proposal) . . . . . 195
9 Residual SAC Johannink et al., 2019 . . . . . 196

# List of Tables


5.1 Average performance ($\pm$ std) over the last $10^5$ timesteps. Comparison of SAC, AVEC-SAC, and EVarEst-SAC ($\alpha^*_{\text{ev}}$: the best performing $\alpha_{\text{ev}}$). † A logging error caused the loss of raw episode returns for the AdroitHandDoor-v1 runs; only relative performance percentages were retained. The reported averages are reconstructed from the baseline score and the relative performance, so no standard deviation is available. . . . . 75

5.2 Average performance ($\pm$ std) over the last $10^5$ timesteps. Comparison of PPO, AVEC-PPO, and EVarEst-PPO ($\alpha^*_{\text{ev}}$: the best performing $\alpha_{\text{ev}}$) . . . . . 75

6.1 JAX Functional Patterns vs. Imperative Python . . . . . 100

7.1 TargetGym environment throughput on CPU (steps/second, averaged over $10^8$ steps after JIT warmup). . . . . 121

8.1 Normalized Expert Advantage (NEA) summary. The last column reports the IQM (inter-quartile mean) of NEA over the last 200k environment steps, computed per seed then aggregated with IQM across seeds. . . . . 146

A.1 Default hyperparameters for both SAC and $EVarEst$-SAC. . . . . 174

A.2 Environment specific hyperparameters for SAC and $EVarEst$-SAC. . . . . 174

A.3 Default hyperparameters for both PPO and $EVarEst$-PPO. . . . . 174

A.4 Environment specific hyperparameters for PPO and $EVarEst$-PPO. . . . . 174

B.1 State variables of the simulation. . . . . 178

B.2 Physical parameters used in the simulation. . . . . 187

C.1 Hyperparameter search space for SAC on `Plane`. . . . . 198

C.2 Tuned SAC hyperparameters (Phase 2 best configuration). . . . . 199

C.3 Fixed SAC hyperparameters (not tuned, shared by all algorithms). . . . . 200

C.4 EDGE hyperparameters (main result: `EDGE _decay_050`). . . . . 201

C.5 Hyperparameters for IBRL and Residual RL baselines. . . . . 201

C.6 IQM permutation test + Holm-Bonferroni for asymptotic performance (IQM over last 200k steps, $\alpha = 0.05$, 10,000 permutations). $p$-values adjusted for multiple comparisons; significant differences ($p^{\text{adj}} < 0.05$) in bold. . . . . 203

C.7 IQM permutation test + Holm-Bonferroni for steps to first reach each NEA threshold (budget-censored, $\alpha = 0.05$, 10,000 permutations). $p$-values adjusted for multiple comparisons; significant differences ($p^{\text{adj}} < 0.05$) in bold. . . . . 204

C.8 Cumulative compute budget. Wall-clock GPU-hours are reported; for the main/ablation/degradation experiments, seed-hours are divided by 100 since seeds were evaluated in batches of 100 in parallel, except for the hyperparameter search (5 seeds in parallel for Phase 1, 15 for Phase 2). . . . . 205

C.9 Cumulative compute budget, estimated energy consumption, and $CO_2$ emissions. All experiments ran on a single server (`scool-bob`) equipped with 4× NVIDIA L40S GPUs (350 W TDP each) and 2× AMD EPYC 9454 CPUs (290 W TDP each). Wall-clock GPU-hours for the main/ablation/degradation experiments are total seed-hours divided by 100 (seeds evaluated in batches of 100 in parallel). Hyperparameter search trials were run sequentially with 5 (Phase 1) or 15 (Phase 2) seeds in parallel per trial. Energy is estimated as wall-clock time × effective system power, assuming 80 % GPU utilisation, 70 % CPU utilisation for batch runs (30–55 % for the search), a system overhead of 150 W, and a PUE of 1.3. $CO_2$ figures use the 2024 French grid intensity of 85 g $CO_2$eq kWh$^{-1}$ (nuclear-heavy); EU and global averages are provided for reference. . . . . 206

# List of References


Abate, Alessandro, Maria Prandini, John Lygeros, and Shankar Sastry (2008). Probabilistic reachability and safety for controlled discrete time stochastic hybrid systems. *Automatica* 44.11, pp. 2724–2734.

Abdolmaleki, Abbas, Jost Tobias Springenberg, Yuval Tassa, Rémi Munos, Nicolas Heess, and Martin A. Riedmiller (2018). Maximum a Posteriori Policy Optimisation. *CoRR* abs/1806.06920.

Adamczyk, Jacob, Volodymyr Makarenko, Stas Tiomkin, and Rahul V Kulkarni (2025). Average-Reward Soft Actor-Critic. In *Reinforcement Learning Conference*.

Advanced Micro Devices (2024). *HIP: Heterogeneous-Compute Interface for Portability*. ROCm Release 6.x.

Agarwal, Rishabh, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare (2021). Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems* 34, pp. 29304–29320.

Airbus S.A.S. (2017). *A320 Flight Crew Training Manual*.

— (2018). *A320 Aircraft Flight Manual*. Aircraft performance and geometric data also available in EASA Type Certificate Data Sheet No. EASA.A.064.

Akiba, Takuya, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Shiba (2019). Optuna: A Next-generation Hyperparameter Optimization Framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2623–2631.

Ali, Matsive, Sandesh Giri, Sen Liu, and Qin Yang (2025). Digital Twin Synchronization: Bridging the Sim-RL Agent to a Real-Time Robotic Additive Manufacturing Control. In *2025 IEEE Engineering Reliable Autonomous Systems (ERAS)*, pp. 1–4.

Anderson, John D. (2016). *Introduction to Flight*. 8th. McGraw-Hill.

Andrychowicz, Marcin, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, et al. (2021). What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study. In *International Conference on Learning Representations (ICLR)*.

Andrychowicz, Marcin, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, et al. (2017). Hindsight Experience Replay. In *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 30.

Ang, Kiam Heong, Gregory Chong, and Yun Li (2005). PID control system analysis, design, and technology. *IEEE Transactions on Control Systems Technology* 13.4, pp. 559–576.

Åström, Karl J. and Tore Hägglund (2001). The future of PID control. *Control Engineering Practice* 9.11, pp. 1163–1175.

Atkinson, Craig, Brendan McCane, Lech Szymanski, and Anthony Robins (Mar. 2021). Pseudo-rehearsal: Achieving deep reinforcement learning without catastrophic forgetting. *Neurocomputing* 428, pp. 291–307.

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton (2016). *Layer Normalization*.

Bach, Francis R., David Heckerman, and Eric Horvitz (2006). Considering Cost Asymmetry in Learning Classifiers. *Journal of Machine Learning Research* 7.63, pp. 1713–1741.

Baier, Christel and Joost-Pieter Katoen (2008). *Principles of Model Checking*. MIT Press.

Balouek, Daniel, Alexandra Carpen Amarie, Ghislain Charrier, Frédéric Desprez, Emmanuel Jeannot, Emmanuel Jeanvoine, et al. (2013). Adding Virtualization Capabilities to the Grid'5000 Testbed. In *Cloud Computing and Services Science*. Ed. by Ivan I. Ivanov, Marten van Sinderen, Frank Leymann, and Tony Shan. Vol. 367. Communications in Computer and Information Science. Springer International Publishing, pp. 3–20.

Barkley, Brett and David Fridovich-Keil (2025a). *A Forensic Analysis of Synthetic Data in RL: Diagnosing and Solving Algorithmic Failures in Model-Based Policy Optimization*. arXiv:2510.01457.

— (2025b). *Stealing That Free Lunch: Exposing the Limits of Dyna-Style Reinforcement Learning*.

Barto, Andrew G., Richard S. Sutton, and Charles W. Anderson (1983). Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems. *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13.5, pp. 834–846.

Beck, Kent (2003). *Test Driven Development: By Example*. Addison-Wesley Professional.

Bellemare, Marc G., Yavar Naddaf, Joel Veness, and Michael Bowling (May 2013). The arcade learning environment: an evaluation platform for general agents. *J. Artif. Int. Res.* 47.1, pp. 253–279.

Bellman, Richard (1957). *Dynamic Programming*. Princeton University Press.

— (Sept. 1958). Dynamic programming and stochastic control processes. *Information and Control* 1.3, pp. 228–239.

— (1984). *Eye of the Hurricane: An Autobiography*. Preface, p. ix. Singapore: World Scientific Publishing Company.

Bellman, Richard E. (1957). Dynamic Programming. In *Science and Cybernetics*. Dynamic programming method that underlies LQR optimal control. Pergamon Press.

Bergstra, James, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl (2011). Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems (NIPS)*. Vol. 24, pp. 2546–2554.

Berkenkamp, Felix, Matteo Turchetta, Angela Schoellig, and Andreas Krause (2017). Safe model-based reinforcement learning with stability guarantees. In *Advances in Neural Information Processing Systems*, pp. 908–918.

Berthelot, Yann (2025a). *Ajax: Reinforcement Learning Agents in Jax*. https://github.com/YannBerthelot/Ajax. Accessed: 2026-03-23.

— (2025b). *TargetGym: Reinforcement Learning Environments for Target MDPs*. Lightweight physics-based RL environments for aircraft, car, and bike control.

Bertsekas, Dimitri and John Tsitsiklis (Jan. 1996). *Neuro-Dynamic Programming*. Vol. 27. Athena Scientific.

Bertsekas, Dimitri P. (2024). Model Predictive Control and Reinforcement Learning: A Unified Framework Based on Dynamic Programming. *IFAC-PapersOnLine* 58.18. 8th IFAC Conference on Nonlinear Model Predictive Control NMPC 2024, pp. 363–383.

Bertsekas, Dimitri P. and Steven E. Shreve (1978). *Stochastic Optimal Control: The Discrete Time Case*. Academic Press.

Bhatt, Aditya, Daniel Palenicek, Boris Belousov, Max Argus, Artemij Amiranashvili, Thomas Brox, et al. (2024). CrossQ: Batch Normalization in Deep Reinforcement Learning for Greater Sample Efficiency and Simplicity. In *International Conference on Learning Representations*.

Biewald, Lukas (2020). *Experiment Tracking with Weights & Biases*. Software available from wandb.com.

Blanchini, Franco (1999). Set invariance in control. *Automatica* 35.11, pp. 1747–1767.

Bloor, Maximilian, Akhil Ahmed, Niki Kotecha, Mehmet Mercangöz, Calvin Tsay, and Ehecactl Antonio Del Rio Chanona (Aug. 2024). *Control-Informed Reinforcement Learning for Chemical Processes*. arXiv:2408.13566 [eess] version: 1.

Bloor, Maximilian, José Torraca, Ilya Orson Sandoval, Akhil Ahmed, Martha White, Mehmet Mercangöz, et al. (2026). PC-Gym: Benchmark environments for process control problems. *Computers & Chemical Engineering* 204, p. 109363.

Bottou, Léon, Frank E. Curtis, and Jorge Nocedal (2018). Optimization Methods for Large-Scale Machine Learning. *SIAM Review* 60.2, pp. 223–311.

Box, George E. P. and Norman R. Draper (1987). *Empirical Model-Building and Response Surfaces*. Wiley Series in Probability and Mathematical Statistics. xiv, 669 p. New York: John Wiley & Sons.

Bradbury, James, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, et al. (2018). *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13.

Brown, Daniel S., Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum (2019). Extrapolating Beyond Suboptimal Demonstrations via Inverse Reinforcement Learning from Observations. *CoRR* abs/1904.06387.

Brunke, Lukas, Michael Greeff, Adam W. Hall, Zijian Yuan, Stephen Zhou, Jacopo Panerati, et al. (2022). Safe learning in robotics: From learning-based control to safe reinforcement learning. *Annual Review of Control, Robotics, and Autonomous Systems* 5, pp. 411–444.

CFM International (2010). *CFM56-5B Product Specification*. Sea-level static thrust for the CFM56-5B4/P: 120.1 kN per engine.

Chang, Ya-Chien, Nima Roohi, and Sicun Gao (2019). Neural Lyapunov Control. In *Advances in Neural Information Processing Systems*. Vol. 32, pp. 3240–3249.

Char, Ian and Jeff Schneider (2023). PID-Inspired Inductive Biases for Deep Reinforcement Learning in Partially Observable Control Tasks. In *Advances in Neural Information Processing*

*Systems*. Ed. by A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine. Vol. 36. Curran Associates, Inc., pp. 59425–59463.

Chen, Xinyue, Che Wang, Zijian Zhou, and Keith W. Ross (2021). *Randomized Ensembled Double Q-Learning: Learning Fast Without a Model*. arXiv:2101.05982.

Chow, Yinlam, Ofir Nachum, Edgar A. Duéñez-Guzmán, and Mohammad Ghavamzadeh (2018). *A Lyapunov-based Approach to Safe Reinforcement Learning*. arXiv:1805.07708.

Citrin, Jonathan, Ian Goodfellow, Akhil Raju, Jeremy Chen, Jonas Degrave, Craig Donner, et al. (2024). *TORAX: A Fast and Differentiable Tokamak Transport Simulator in JAX*.

Coulom, Rémi (2006). Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In *Proceedings of the 5th International Conference on Computers and Games (CG 2006)*. Turin, Italy: Springer, pp. 72–83.

Cuoq, Pascal, Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, and Boris Yakobowski (2015). Frama-C: A Software Analysis Perspective. *Formal Aspects of Computing* 27.3, pp. 573–609.

D'Eramo, Carlo, Davide Tateo, Andrea Bonarini, Marcello Restelli, and Jan Peters (2020). MushroomRL: Simplifying Reinforcement Learning Research. *Journal of Machine Learning Research* 21.131, pp. 1–5.

Degrave, Jonas, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, et al. (Feb. 2022). Magnetic control of tokamak plasmas through deep reinforcement learning. en. *Nature* 602.7897. Publisher: Nature Publishing Group.

Deng, Jifei, Seppo Sierla, Jie Sun, and Valeriy Vyatkin (June 2023). Offline reinforcement learning for industrial process control: A case study from steel industry. en. *Information Sciences* 632, pp. 221–231.

Dhakal, Shobhakar, Jan C. Minx, Ferenc L. Toth, Anas Abdel-Aziz, Maria Josefina Figueroa Meza, Klaus Hubacek, et al. (2022). Emissions Trends and Drivers. In *Climate Change 2022: Mitigation of Climate Change. Contribution of Working Group III to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change*. Ed. by P.R. Shukla, J. Skea, R. Slade, A. Al Khourdajie, R. van Diemen, D. McCollum, et al. Cambridge University Press. Chap. 2.

Dogru, O., M. Gürbüz, M. Yüksel, and M. Köksal (2024). Reinforcement Learning in Process Industries: Review and Perspective. *IEEE Transactions on Automation Science and Engineering* 21.4, pp. 1743–1755.

Dogru, Oguzhan, Kirubakaran Velswamy, Fadi Ibrahim, Yuqi Wu, Arun Senthil Sundaramoorthy, Biao Huang, et al. (2022). Reinforcement learning approach to autonomous PID tuning. *Computers & Chemical Engineering* 161, p. 107760.

Dohare, Shibhansh, J. Fernando Hernandez-Garcia, Qingfeng Lan, Parash Rahman, A. Rupam Mahmood, and Richard S. Sutton (Aug. 2024). Loss of plasticity in deep continual learning. en. *Nature* 632.8026, pp. 768–774.

Dulac-Arnold, Gabriel, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, et al. (2021). Challenges of Real-World Reinforcement Learning: Definitions, Benchmarks and Analysis. *Machine Learning* 110.9, pp. 2419–2468.

Engstrom, Logan, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, et al. (2020). Implementation Matters in Deep Policy Gradients: A Case Study on PPO and TRPO. In *International Conference on Learning Representations* (*ICLR*).

Ernst, Damien, Pierre Geurts, and Louis Wehenkel (2005). Tree-Based Batch Mode Reinforcement Learning. *Journal of Machine Learning Research* 6, pp. 503–556.

Espeholt, Lasse, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, et al. (2018). IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. In *International Conference on Machine Learning*. PMLR, pp. 1407–1416.

Eustace, Sébastien et al. (2018). *Poetry: Python packaging and dependency management made easy*. Version 1.8.0.

Farahmand, Amir-Massoud, Andre Barreto, and Daniel Nikovski (20–22 Apr 2017). Value-Aware Loss Function for Model-based Reinforcement Learning. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. Ed. by Aarti Singh and Jerry Zhu. Vol. 54. Proceedings of Machine Learning Research. PMLR, pp. 1486–1494.

Federal Aviation Administration (2004). *Advisory Circular AC 25.143-1: Handling Qualities*. Tech. rep. FAA.

Filliâtre, Jean-Christophe and Andrei Paskevich (2013). Why3: Where Programs Meet Provers. In *Programming Languages and Systems* (*ESOP*). Vol. 7792. Lecture Notes in Computer Science. Springer, pp. 125–128.

Flet-Berliac, Yannis, Reda Ouhamma, Odalric-Ambrym Maillard, and Philippe Preux (2021). Learning Value Functions in Deep Policy Gradients using Residual Variance. In *Proc. ICLR*.

Forbes, Michael G., Rohit S. Patwardhan, Hamza Hamadah, and R. Bhushan Gopaluni (2015). Model Predictive Control in Industry: Challenges and Opportunities. en. *IFAC-PapersOnLine* 48.8, pp. 531–538.

Freeman, C. Daniel, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem (2021). Brax - A Differentiable Physics Engine for Large Scale Rigid Body Simulation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track* (*Round 1*).

Fu, Jie and Ufuk Topcu (2014). Probably Approximately Correct MDP Learning and Control with Temporal Logic Constraints. In *Robotics: Science and Systems*.

Fujimoto, Scott, Herke van Hoof, and David Meger (2018). Addressing Function Approximation Error in Actor-Critic Methods. In *International Conference on Machine Learning* (*ICML*).

Fujimoto, Scott, David Meger, and Doina Precup (2019). Off-Policy Deep Reinforcement Learning without Exploration. In *International Conference on Machine Learning*, pp. 2052–2062.

Gallici, Matteo, Mattie Fellows, Benjamin Ellis, Bartomeu Pou, Ivan Masmitja, Jakob Nicolaus Foerster, et al. (2025). *Simplifying Deep Temporal Difference Learning*.

García, C. E., D. M. Prett, and M. Morari (1989). Model Predictive Control: Theory and Practice — A Survey. *Automatica* 25.3, pp. 335–348.

Geman, Stuart, Elie Bienenstock, and René Doursat (1992). Neural Networks and the Bias/-Variance Dilemma. *Neural Computation* 4.1, pp. 1–58.

Ha, David and Jürgen Schmidhuber (2018). *Recurrent World Models Facilitate Policy Evolution*. arXiv:1809.01999.

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, and Sergey Levine (2018). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *Proceedings of the 35th International Conference on Machine Learning*. PMLR, pp. 1861–1870.

Haarnoja, Tuomas, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, et al. (2019). *Soft Actor-Critic Algorithms and Applications*.

Hafner, Danijar, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi (2020). Dream to Control: Learning Behaviors by Latent Imagination. In *International Conference on Learning Representations*.

Hafner, Danijar, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, et al. (2019). Learning Latent Dynamics for Planning from Pixels. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR, pp. 2555–2565.

Hafner, Danijar, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba (2021). Mastering Atari with Discrete World Models. In *International Conference on Learning Representations*.

Hafner, Danijar, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap (2025). Mastering Diverse Control Tasks through World Models. *Nature*, pp. 1–7.

Hairer, Ernst, Syvert Paul Nørsett, and Gerhard Wanner (1993). *Solving Ordinary Differential Equations I: Nonstiff Problems*. 2nd. Springer.

Hampel, Frank R. (Dec. 1971). A General Qualitative Definition of Robustness. *The Annals of Mathematical Statistics* 42.6. Publisher: Institute of Mathematical Statistics, pp. 1887–1896.

Hansen, Nicklas A, Hao Su, and Xiaolong Wang (17–23 Jul 2022). Temporal Difference Learning for Model Predictive Control. In *Proceedings of the 39th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato. Vol. 162. Proceedings of Machine Learning Research. PMLR, pp. 8387–8406.

— (2024). TD-MPC2: Scalable, Robust World Models for Continuous Control. In *International Conference on Learning Representations (ICLR)*.

Harris, Charles R., K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, et al. (Sept. 2020). Array programming with NumPy. *Nature* 585.7825, pp. 357–362.

Hasselt, Hado van, Arthur Guez, and David Silver (2016). Deep Reinforcement Learning with Double Q-Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 1, pp. 2094–2100.

Heek, Jonathan, Anselm Levskaya, Avital Oliver, Marvin Ritter, Adrien Bertrand, Andreas Müller, et al. (2020). *Flax: A neural network library and ecosystem for JAX designed for flexibility*. Version 0.7.2.

Henderson, Peter, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger (2018). Deep Reinforcement Learning that Matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1.

Hernández-Lerma, Onésimo and Jean Bernard Lasserre (2012). *Discrete-time Markov control processes: Basic optimality criteria*. Springer Science & Business Media.

Hessel, Matteo, David Budden, Gabriel Barth-Maron, Hado van Hasselt, Joseph Modayil, and Albin Cassirer (2020). *Optax: Composable gradient transformation and optimization, in JAX*. Version 0.1.7.

Hessel, Matteo, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, et al. (2018). Rainbow: Combining Improvements in Deep Reinforcement Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1.

Hiraoka, Takuya, Takahisa Imagawa, Taisei Hashimoto, Takashi Onishi, and Yoshimasa Tsuruoka (2021). *Dropout Q-Functions for Doubly Efficient Reinforcement Learning*. arXiv:2110.02034.

Ho, Jonathan and Stefano Ermon (2016). Generative Adversarial Imitation Learning. In *Advances in Neural Information Processing Systems*. Vol. 29, pp. 4565–4573.

Hoang, H., Z. Li, M. Zhang, and D. A. Bagnell (2024). Sub-optimal Demonstrations driven Offline Imitation Learning. In *Proceedings of the 38th Conference on Neural Information Processing Systems (NeurIPS)*.

Howard, Ronald A. (1960). *Dynamic Programming and Markov Processes*. Cambridge, MA, USA: Technology Press of Massachusetts Institute of Technology.

Hu, Hengyuan, Suvir Mirchandani, and Dorsa Sadigh (2023). Imitation Bootstrapped Reinforcement Learning. *arXiv preprint arXiv:2311.02198*.

Huang, Shenanyi, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, et al. (2022). CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms. *Journal of Machine Learning Research* 23.274, pp. 1–18.

Huang, Shengyi, Rousslan Fernand Julien Dossa, Antonin Raffin, Anssi Kanervisto, and Weixun Wang (2022). The 37 Implementation Details of Proximal Policy Optimization. In *ICLR Blog Track*. Accessed: 2024-03-21.

Hussein, Ahmed, Mohamed Medhat Gaber, Eyad Elyan, and Chris Jayne (2017). Imitation Learning: A Survey of Learning Methods. *ACM Computing Surveys (CSUR)* 50.2, pp. 1–35.

Igl, Maximilian, Gregory Farquhar, Jelena Luketina, Wendelin Boehmer, and Shimon Whiteson (2021). Transient Non-Stationarity and Generalisation in Deep Reinforcement Learning. In *International Conference on Learning Representations*.

Ilyas, Andrew, Logan Engstrom, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, et al. (2020). A Closer Look at Deep Policy Gradients. In *Proc. ICLR*.

International Organization for Standardization (1975). *Standard Atmosphere*. Tech. rep. ISO 2533:1975. ISO.

Islam, Riashat, Peter Henderson, Ahmed Mazumdar, Marc G. Hoffmann, and Joelle Pineau (2017). Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control. In *ICML Workshop on Reproducibility in Machine Learning*.

ISO (2020). *ISO/IEC 14882:2020 Information technology — Programming languages — C++*. Geneva, Switzerland: International Organization for Standardization.

Jackson, Paul, ed. (n.d.). *Jane's All the World's Aircraft*. Various years. IHS Jane's.

Jang, Eric, Shixiang Gu, and Ben Poole (2017). Categorical Reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations*.

Janner, Michael, Justin Fu, Marvin Zhang, and Sergey Levine (2019). *When to Trust Your Model: Model-Based Policy Optimization*. arXiv:1906.08253.

Johannink, Tobias, Shikhar Bahl, Ashvin Yeracker, Saurabh Sahai, Avinash Kumar, Sergey Levine, et al. (2019). Residual Reinforcement Learning for Robot Control. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 6023–6029.

Jones, Andy (Jan. 2021). *Debugging Reinforcement Learning Systems*. Blog post. Accessed: 2024-03-21.

Kaelbling, Leslie Pack, Michael L. Littman, and Anthony R. Cassandra (1998). Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence* 101.1–2, pp. 99–134.

Kalashnikov, Dmitry, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, et al. (2018). Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation. In *Proceedings of The 2nd Conference on Robot Learning*. Vol. 87. PMLR, pp. 651–673.

Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME – Journal of Basic Engineering* 82. Introduced Kalman filter and fundamental state-space optimal control concepts underlying the LQR, pp. 35–45.

Karten, Seth, Rahul Dev Appapogu, and Chi Jin (2026). *Automatic Generation of High-Performance RL Environments*.

Kearns, Michael and Satinder Singh (1998). Finite-Sample Convergence Rates for Q-Learning and Indirect Algorithms. In *Advances in Neural Information Processing Systems*. Ed. by M. Kearns, S. Solla, and D. Cohn. Vol. 11. MIT Press.

Kocsis, Levente and Csaba Szepesvári (2006). Bandit Based Monte-Carlo Planning. In *Proceedings of the Seventeenth European Conference on Machine Learning (ECML 2006)*. Vol. 4212. Lecture Notes in Computer Science. Berlin/Heidelberg, Germany: Springer, pp. 282–293.

Kolmanovsky, Ilya and Elmer G. Gilbert (1998). The theory and computation of disturbance invariant sets for discrete-time linear systems. *Mathematical Problems in Engineering* 4.4, pp. 317–367.

Kondo, Sebastian, John D. Co-Reyes, Dale Schuurmans, and Sergey Levine (2020). *Revisiting Rainbow: Promoting Speculative Research Through Code Backporting*. arXiv:2011.14826.

Konishi, Mikito, Masanobu Inubushi, and Susumu Goto (Aug. 2022). Fluid mixing optimization with reinforcement learning. en. *Scientific Reports* 12.1, p. 14268.

Kress-Gazit, Hadas, Georgios E. Fainekos, and George J. Pappas (2009). Temporal-logic-based reactive mission and motion planning. In *IEEE International Conference on Robotics and Automation*. IEEE, pp. 3116–3121.

Kuhnle, Andreas and Gisela Lanza (2019). Application of Reinforcement Learning in Production Planning and Control of Cyber Physical Production Systems. In *Machine Learning for Cyber Physical Systems*. Ed. by Jürgen Beyerer, Christian Kühnert, and Oliver Niggemann. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 123–132.

Kumar, Aviral, Rishabh Agarwal, Dibya Ghosh, and Sergey Levine (2021). Implicit Under-Parameterization Inhibits Data-Efficient Deep Reinforcement Learning. In *Proc. ICLR*.

Kumar, Aviral, Rishabh Agarwal, Tengyu Ma, Aaron Courville, George Tucker, and Sergey Levine (2022). DR3: Value-Based Deep Reinforcement Learning Requires Explicit Regularization. In *International Conference on Learning Representations*.

Lange, Robert Tjarko (2022). *gymnax: A JAX-based Reinforcement Learning Environment Library*. Version 0.0.4.

Lattner, Chris and Vikram Adve (2004). LLVM: A Compilation Framework for Lifelong Program Analysis and Transformation. In *Proceedings of the International Symposium on Code Generation and Optimization (CGO)*. San Jose, CA, USA, pp. 75–86.

Leary, Chris and Todd Wang (2017). *XLA: Accelerated Linear Algebra*. Google Developers Blog.

Lehmann, Erich L. and Joseph P. Romano (2022). *Testing Statistical Hypotheses*. Springer Texts in Statistics. Springer Cham.

Lehtosalo, Jukka et al. (2012). *Mypy: Optional static typing for Python*.

Li, Z. (2022). How to Improve Imitation Learning Performance with Sub-optimal Demonstrations. *Proceedings of the 2022 ICML Workshop on Distributional Reinforcement Learning*.

Li, Zechu, Tao Chen, Zhang-Wei Hong, Anurag Ajay, and Pulkit Agrawal (2023). *Parallel Q-Learning: Scaling Off-policy Reinforcement Learning under Massively Parallel Simulation*.

Liang, Eric, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Joseph Gonzalez, et al. (2018). RLlib: Abstractions for Distributed Reinforcement Learning. In *International Conference on Machine Learning (ICML)*, pp. 3053–3062.

Lillicrap, Timothy P., Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, et al. (2016). Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun.

Lyapunov, A. M. (1992). The general problem of the stability of motion. *International Journal of Control* 55.3, pp. 531–534.

Lyle, Clare, Mark Rowland, and Will Dabney (2022). Understanding and Preventing Capacity Loss in Reinforcement Learning. In *Proc. ICLR*.

Ma, Xiaoteng, Xiaohang Tang, Li Xia, Jun Yang, and Qianchuan Zhao (2021). *Average-Reward Reinforcement Learning with Trust Region Methods*. arXiv:2106.03442.

Maas, Andrew L., Awni Y. Hannun, and Andrew Y. Ng (2013). Rectifier Nonlinearities Improve Neural Network Acoustic Models. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*. Vol. 30. 1, p. 3.

Maddison, Chris J., Andriy Mnih, and Yee Whye Teh (2017). The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *International Conference on Learning Representations*.

Makoviychuk, Viktor, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, et al. (2021). Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Mark, Christoph and Steven Liu (2020). Stochastic MPC with Distributionally Robust Chance Constraints. *IFAC-PapersOnLine* 53.2. 21st IFAC World Congress, pp. 7136–7141.

Marsh, Charlie and Astral Software (2023). *Ruff: An extremely fast Python linter and code formatter*. Version 0.3.0.

Maurer, Andreas and Massimiliano Pontil (2009). Empirical Bernstein Bounds and Sample-Variance Penalization. In *COLT*.

McCloskey, Michael and Neal J. Cohen (Jan. 1989). Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. In *Psychology of Learning and Motivation*. Ed. by Gordon H. Bower. Vol. 24. Academic Press, pp. 109–165.

Mendiola-Rodriguez, Tannia A. and Luis A. Ricardez-Sandoval (June 2022). Robust control for anaerobic digestion systems of Tequila vinasses under uncertainty: A Deep Deterministic Policy Gradient Algorithm. *Digital Chemical Engineering* 3, p. 100023.

Micheli, Vincent, Eloi Alonso, and François Fleuret (2023). Transformers are Sample-Efficient World Models. In *The Eleventh International Conference on Learning Representations, ICLR 2023*.

Mnih, Volodymyr, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, et al. (20–22 Jun 2016). Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, pp. 1928–1937.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, et al. (2013). *Playing Atari with Deep Reinforcement Learning*. arXiv:1312.5602.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, et al. (2015). Human-Level Control through Deep Reinforcement Learning. *Nature* 518.7540, pp. 529–533.

Moerland, Thomas M., Joost Broekens, and Catholijn M. Jonker (2020). *Model-based Reinforcement Learning: A Survey*. arXiv:2006.16712.

Mouchamps, Antoine, Arthur Malherbe, Adrien Bolland, and Damien Ernst (2025). *GymTORAX: Open-source software for integrating RL with plasma control simulators*.

Munroe, Randall (Aug. 2020). *Dependency*. xkcd. Accessed: 2026-03-16.

Nair, Arun, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, et al. (2015). Massively Parallel Methods for Deep Reinforcement Learning. *arXiv preprint arXiv:1507.04296*.

Nair, Vinod and Geoffrey E. Hinton (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pp. 807–814.

Nikishin, Evgenii, Junhyuk Oh, Georg Ostrovski, Clare Lyle, Razvan Pascanu, Will Dabney, et al. (2023). Deep Reinforcement Learning with Plasticity Injection. In *Advances in Neural Information Processing Systems*. Vol. 36, pp. 37142–37159.

Nikishin, Evgenii, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville (June 2022). The Primacy Bias in Deep Reinforcement Learning. en. In *Proc. ICML*. PMLR, pp. 16828–16847.

NVIDIA Corporation (2024). *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*. Version 12.x.

Osa, Takayuki, Fabio Pardo, Gerhard Neumann, Hiroshi Ishiguro, and Jan Peters (2018). An Algorithmic Perspective on Imitation Learning. *Foundations and Trends in Robotics* 7.1–2, pp. 1–179.

Osband, Ian, Yotam Doron, Matteo Hessel, John Aslanides, Eren Sezener, Andre Saraiva, et al. (Sept. 2019). Behaviour Suite for Reinforcement Learning. en. In

Oshin, Alex, Hassan Almubarak, and Evangelos A. Theodorou (2024). *Differentiable Robust Model Predictive Control*.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems* 32.

Peng, Baolin, Xiujun Li, Jianfeng Gao, Jingjing Liu, and Kam-Fai Wong (2018). *Integrating planning for task-completion dialogue policy learning*. arXiv:1801.06176.

Peng, Xue Bin, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel (2018). Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 3803–3810.

Pilkington, Lionel Alexander Bethune (1959). Manufacture of Flat Glass. GB769692. Original patent describing the float glass process.

Pineau, Joelle, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Larivière, Adrien Rousseau, Pascal Vincent, et al. (2021). The Machine Learning Reproducibility Checklist. *Journal of Machine Learning Research* 22.1, pp. 1–20.

Pourshamsaei, Hossein and Amin Nobakhti (2024). Predictive reinforcement learning in non-stationary environments using weighted mixture policy. *Applied Soft Computing* 153, p. 111305.

Powell, By Kody M., Derek Machalek, and Titus Quah (2020). Real-time optimization using reinforcement learning. *Computers & Chemical Engineering* 143, p. 107077.

Puterman, Martin L (1994). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Qin, S. Joe and Thomas A. Badgwell (2003). A survey of industrial model predictive control technology. *Control Engineering Practice* 11.7, pp. 733–764.

Radji, Waris, Thomas Michel, and Hector Piteau (2026). Octax: Accelerated CHIP-8 Arcade Environments for Reinforcement Learning in JAX. In *The Fourteenth International Conference on Learning Representations*.

Raffin, Antonin (2020). *RL Baselines3 Zoo*. https://github.com/DLR-RM/rl-baselines3-zoo.

— (Feb. 2025). *Getting SAC to Work on a Massive Parallel Simulator: An RL Journey With Off-Policy Algorithms (Part I)*. en-us.

Raffin, Antonin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann (2021). Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research* 22.268, pp. 1–8.

Raj, Anant, Pierre Gaillard, and Christophe Saad (Nov. 2020). *Non-stationary Online Regression*. Not published so only on arXiv/HAL.

Rajeswaran, Aravind, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, et al. (2018). Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. In *Robotics: Science and Systems (RSS)*.

Ramstedt, Simon and Christopher Pal (2019). Real-Time Reinforcement Learning. In *Advances in Neural Information Processing Systems* (*NeurIPS*). Vol. 32.

Randløv, Jette and Preben Alstrøm (1998). Learning to Drive a Bicycle using Reinforcement Learning and Shaping. In *ICML*. Vol. 98, pp. 463–471.

Raymer, Daniel P. (2018). *Aircraft Design: A Conceptual Approach*. 6th. AIAA Education Series.

Reid, Mark D. and Robert C. Williamson (Dec. 2009). *Composite Binary Losses*. arXiv:0912.3301 [stat].

Richalet, J., A. Rault, J.L. Testud, and J. Papon (1978). Model predictive heuristic control: Applications to industrial processes. *Automatica* 14.5, pp. 413–428.

RLcore (2024a). *Optimizing Water Treatment Ultrafiltration with Reinforcement Learning*. https://drive.google.com/file/d/1-lctkJRYpnNpRB_vkg_VKhVBLMj_CUoR/view. Technical report.

— (2024b). *Reducing Wastewater Treatment Emissions with Reinforcement Learning: H2S Scrubber Efficiency*. https://drive.google.com/file/d/1UpjTcpkiXF-jwZniPMdKyIWRmaUlldsl/view. Technical report.

— (2026). *RLcore – Intelligent Control with Reinforcement Learning*. https://rlcore.ai. Accessed: 2026-03-31.

Roskam, Jan (2001). *Airplane Flight Dynamics and Automatic Flight Controls*. Vol. I. DARcorporation.

Ross, Stéphane, Geoffrey Gordon, and Drew Bagnell (2011). A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 627–635.

Rouxel, Jacques (1968). *Et voilà le Shadok : Season 1, Episode 27*. Originally broadcast 1968. INA digital archive accessed via YouTube at 01:27.

— (1968–1974). *Les Shadoks*. Série d'animation, ORTF. Aphorisme : « En essayant continuellement, on finit par réussir… Donc plus ça rate, plus on a de chances que ça marche. » France.

Rubinstein, Reuven (1999). The Cross-Entropy Method for Combinatorial and Continuous Optimization. *Methodology and Computing in Applied Probability* 1.2, pp. 127–190.

Rubinstein, Reuven Y and Dirk P Kroese (2004). *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning*. New York: Springer.

Rudin, Nikita, David Hoeller, Philipp Reist, and Marco Hutter (Aug. 2022). Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning. In *Proceedings of the 5th Conference on Robot Learning*. Ed. by Aleksandra Faust, David Hsu, and Gerhard Neumann. Vol. 164. Proceedings of Machine Learning Research. PMLR, pp. 91–100.

Saint-Exupéry, Antoine de (1939). *Terre des hommes*. English title: *Wind, Sand and Stars*. Paris: Gallimard.

Saint-Gobain (Jan. 2021). *Saint-Gobain Glass - Float Process*. https://www.youtube.com/watch?v=1HDWJgFLCfA.

Salimans, Tim, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever (2017). Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *arXiv preprint arXiv:1703.03864*.

Schegg, P., E. Ménager, E. Khairallah, D. Marchal, J. Dequidt, Ph. Preux, et al. (Apr. 2023). SofaGym: An open platform for Machine Learning based on Soft Robot simulations. *Soft Robotics* 10, pp. 410–430.

Schoettler, Gerrit, Ashvin Nair, Jianlan Luo, Shikhar Bahl, Juan Aparicio Ojea, Eugen Solowjow, et al. (2020). Deep Reinforcement Learning for Industrial Insertion Tasks with Visual Inputs and Natural Rewards. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5548–5555.

Schrittwieser, Julian, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, et al. (2020). Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. *Nature* 588.7839, pp. 604–609.

Schulman, John, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz (July 2015). Trust Region Policy Optimization. In *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, pp. 1889–1897.

Schulman, John, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel (Oct. 2018). *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. arXiv:1506.02438 [cs].

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (Aug. 2017). *Proximal Policy Optimization Algorithms*. arXiv:1707.06347.

Schwartz, Roy, Jesse Dodge, Noah A. Smith, and Oren Etzioni (2020). Green AI. *Communications of the ACM* 63.12, pp. 54–63.

Sculley, D., Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, et al. (2014). Machine Learning: The High-Interest Credit Card of Technical Debt. In *NIPS'14 Workshop on Software Engineering for Machine Learning (SE4ML)*. Google, Inc.

Seborg, Dale E, Thomas F Edgar, Duncan A Mellichamp, and Francis J Doyle (2016). *Process Dynamics and Control*. 4th. Hoboken, NJ: Wiley.

Seo, Younggyo, Carmelo Sferrazza, Juyue Chen, Guanya Shi, Rocky Duan, and Pieter Abbeel (2025). *Learning Sim-to-Real Humanoid Locomotion in 15 Minutes*.

Seo, Younggyo, Carmelo Sferrazza, Haoran Geng, Michal Nauman, Zhao-Heng Yin, and Pieter Abbeel (2025). *FastTD3: Simple, Fast, and Capable Reinforcement Learning for Humanoid Control*.

Sharma, Vipul Kumar, Wesley A. Suttle, and S. Sivaranjani (Oct. 2025). Globally Optimal Policy Gradient Algorithms for Reinforcement Learning with PID Control Policies. en. In

Shukla, Arth (Mar. 2025). *Speeding Up SAC with Massively Parallel Simulation*. Substack newsletter.

Sikchi, Harshit, Wenxuan Zhou, and David Held (2021). LOOP: Learning Off-Policy with Online Planning. In *5th Annual Conference on Robot Learning*.

Silver, David, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, et al. (2016). Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature* 529.7587, pp. 484–489.

Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, et al. (2018). A General Reinforcement Learning Algorithm that Masters Chess, Shogi, and Go Through Self-Play. *Science* 362.6419, pp. 1140–1144.

Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, et al. (2017). Mastering the Game of Go Without Human Knowledge. *Nature* 550.7676, pp. 354–359.

Silver, Tom, Kelsey Allen, Stephanie Lew, Leslie Pack Kaelbling, and Tomas Lozano-Pérez (2018). Residual Policy Learning. In *arXiv preprint arXiv:1812.06298*.

Sokar, Ghada, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci (2023). The Dormant Neuron Phenomenon in Deep Reinforcement Learning. In *Proceedings of the 40th International Conference on Machine Learning*. PMLR, pp. 32145–32168.

Strehl, Alexander L. and Michael L. Littman (2008). An analysis of model-based Interval Estimation for Markov Decision Processes. *Journal of Computer and System Sciences* 74.8. Learning Theory 2005, pp. 1309–1331.

Sutton, Richard S. (1988). Learning to Predict by the Methods of Temporal Differences. *Machine Learning* 3, pp. 9–44.

— (July 1991). Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.* 2.4, pp. 160–163.

Sutton, Richard S. and Andrew Barto (2020). *Reinforcement learning: an introduction*. en. Second. Adaptive computation and machine learning. Cambridge, Massachusetts London, England: The MIT Press.

Swiechowski, Maciej, Konrad Godlewski, Bartosz Sawicki, and Jacek Mandziuk (2021). *Monte Carlo Tree Search: A Review of Recent Modifications and Applications*. arXiv:2103.04931.

Tang, Chen, Ben Abbatematteo, Jiaheng Hu, Rohan Chandra, Roberto Martín-Martín, and Peter Stone (2024). Deep Reinforcement Learning for Robotics: A Survey of Real-World Successes. *Annual Review of Control, Robotics, and Autonomous Systems*.

The Coq Development Team (2024). *The Coq Proof Assistant Reference Manual*. INRIA.

Tobin, Josh, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 23–30.

Todorov, Emanuel, Tom Erez, and Yuval Tassa (2012). MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 5026–5033.

Torenbeek, Egbert (1982). *Synthesis of Subsonic Airplane Design*. Delft University Press / Kluwer.

Towers, Mark, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, et al. (2024). *Gymnasium: A Standard Interface for Reinforcement Learning Environments*. arXiv preprint arXiv:2407.17032.

Tukey, John W. (1962). The Future of Data Analysis. *The Annals of Mathematical Statistics* 33.1, pp. 1–67.

Tunyasuvunakool, Saran, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, et al. (2020). dm_control: Software and tasks for continuous control. *Software Impacts* 6, p. 100022.

van Niekerk, J.A., J.D. le Roux, and I.K. Craig (2025). Reinforcement learning based automatic tuning of PID controllers in multivariable grinding mill circuits. *Control Engineering Practice* 165, p. 106522.

Van Rossum, Guido and Fred L. Drake (2009). *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace.

Wang, Liyuan, Xingxing Zhang, Hang Su, and Jun Zhu (2024). A Comprehensive Survey of Continual Learning: Theory, Method and Application. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46.8, pp. 5362–5383.

Watkins, Christopher J. C. H. (1989). Learning from Delayed Rewards. PhD thesis. King's College, University of Cambridge.

Watkins, Christopher J. C. H. and Peter Dayan (May 1992). Q-learning. en. *Machine Learning* 8.3, pp. 279–292.

Weng, Jiayi, Min Lin, Shengyi Huang, Bo Liu, Denys Makoviichuk, Viktor Makoviychuk, et al. (2022). EnvPool: A Highly Parallel Reinforcement Learning Environment Execution Engine. In *Advances in Neural Information Processing Systems*. Vol. 35, pp. 22409–22421.

Werbos, Paul J. (1990). Backpropagation Through Time: What It Does and How to Do It. *Proceedings of the IEEE* 78.10, pp. 1550–1560.

William of Conches (c. 1123). Glosae super Priscianum. In *Guillelmi de Conchis Glosae super Priscianum*. Ed. by Édouard Jeauneau. Vol. 1501. Corpus Christianorum Continuatio Mediaevalis. Modern edition published in 2009. Original work circa 1123. Turnhout: Brepols.

Williams, Grady, Paul Drews, Brian Goldfain, James M. Rehg, and Evangelos A. Theodorou (2016). Aggressive driving with model predictive path integral control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1433–1440.

Williams, Ronald J. (May 1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. en. *Machine Learning* 8.3, pp. 229–256.

Witman, Matthew, Dogan Gidon, David B Graves, Berend Smit, and Ali Mesbah (Sept. 2019). Sim-to-real transfer reinforcement learning for control of thermal effects of an atmospheric pressure plasma jet. en. *Plasma Sources Science and Technology* 28.9. Publisher: IOP Publishing, p. 095019.

Wu, Yuhuai, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba (2017). Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. In *Advances in neural information processing systems*, pp. 5279–5288.

Wu, Yuhuai, Elman Mansimov, Shun Liao, Alec Radford, and John Schulman (Aug. 2017). *OpenAI Baselines: ACKTR & A2C*. https://openai.com/research/openai-baselines-acktr-a2c. Archived at: https://web.archive.org/web/20230327182809/https://openai.com/research/openai-baselines-acktr-a2c.

Wu, Yuting, Lantao Xing, Fanghong Guo, and Xiaokang Liu (2021). On the Combination of PID control and Reinforcement Learning: A Case Study with Water Tank System. In *2021 IEEE 16th Conference on Industrial Electronics and Applications (ICIEA)*, pp. 1877–1882.

Ye, Weirui, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao (2021). Mastering Atari Games with Limited Data. In *Advances in Neural Information Processing Systems*. Vol. 34, pp. 25476–25488.

Young, Kenny and Tian Tian (2019). *MinAtar: An Atari-Inspired Testbed for Thorough and Reproducible Reinforcement Learning Experiments*.

Yu, Xinyi, Siyu Xu, Yuehai Fan, and Linlin Ou (Dec. 2025). Self-Adaptive LSAC-PID Approach Based on Lyapunov Reward Shaping for Mobile Robots. en. *Journal of Shanghai Jiaotong University (Science)* 30.6, pp. 1085–1102.

Zakka, Kevin, Baruch Tabanpour, Qiayuan Liao, Mustafa Haiderbhai, Samuel Holt, Jing Yuan Luo, et al. (2025). *MuJoCo Playground: An open-source framework for GPU-accelerated robot learning and sim-to-real transfer.*

Zhang, Bin, Weihao Hu, Di Cao, Qi Huang, Zhe Chen, and Frede Blaabjerg (2019). Deep reinforcement learning–based approach for optimizing energy conversion in integrated electrical and heating system with renewable energy. *Energy Conversion and Management* 202, p. 112199.

Zhao, H., Q. Gu, Y. Xingrui, D. Bossens, and I. Tsang (2025). Beyond-Expert Performance with Limited Demonstrations. *arXiv preprint arXiv:2506.20307*.

Ziegler, J. G. and N. B. Nichols (1942). Optimum Settings for Automatic Controllers. *Transactions of the American Society of Mechanical Engineers* 64.8, pp. 759–765.