# Leveraging Constraint Space Manifolds and Lottery Tickets to Learn Complex Control Tasks with Pruned Neural Networks

by

Rushil Kumar

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for the degree of
Master of Applied Science
in
Electrical and Computer Engineering with a concentration in A.I.

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

**Examining Committee**

The following served on the Examining Committee for this thesis.

Carleton Member: Peter X. Liu
Professor, Department of Systems and Computer Engineering
Carleton Univeristy

Internal Member(s): Davide Spinello
Professor, Department of Mechanical Engineering
University of Ottawa

Supervisor(s): Wail Gueaieb
Professor, School of Electrical Engineering & Computer Science
University of Ottawa, Canada

## Declaration of Authorship

I hereby certify that this thesis is entirely my own original work except where otherwise indicated. I am aware of the University of Ottawa regulations concerning plagiarism, including those regarding consequent disciplinary actions. Any use of the works of any other author, in any form, is properly acknowledged at their point of use.

## Abstract

Path-planning and control tasks have always been quintessential problems in Robotics. The application of Deep Reinforcement Learning (Deep-RL) to these problems has enabled far more dynamic and adaptive solutions than traditional model-based approaches. In recent years, considerable research attention has shifted towards safe reinforcement learning, where uncertainty and safety constraints are involved directly into the learning process. These methods ensure that agents learn policies that avoid unsafe behaviors, such as collisions, while still exploring effectively. However, for complex, high-dimensional control tasks, the learning and deployment pipeline become increasingly burdensome, as Deep-RL models require substantial computational resources. Sparse and pruned networks offer potential solutions for deployment efficiency. Nonetheless, training such networks from scratch remains challenging due to limited representational capacity.

Learning highly complex tasks with a sparse model from the outset is particularly difficult, as the reduced parameter count restricts the model's ability to explore, often resulting in unstable learning dynamics. This issue is pronounced for off-policy actor-critic algorithms, which rely heavily on stable function approximation. Conversely, pruning a fully trained dense model may appear to be a viable alternative. However, doing so risks removing parameters that encode essential behaviors or safety-related information. Such indiscriminate loss can lead to unpredictable or unsafe policies, especially in environments governed by physical constraints, where certain safety conditions may be implicitly encoded in specific network weights.

In this work, we propose a research hypothesis (based on the Lottery Ticket Hypothesis (LTH)) that guides the development of an off-policy reinforcement learning framework, which will call *Trikaya*; designed to robustly train a shortened network (with significantly fewer active nodes) from scratch while preserving physical safety throughout learning. By leveraging the tangent space of the environment's Constraint Space Manifold (CSM) (via ATACOM environments), the framework ensures that all learnable actions adhere to safety constraints inherent to the environment. Lottery ticket sub-networks are extracted from a fully trained dense model using Iterative Magnitude Pruning (IMP) or One Shot Magnitude Pruning (OSMP). These masks are then applied for the reinitialization step of the LTH for building the shortened network, which after training, can accomplish tasks with similar metrics as the full dense network while maintaining safe learning steps. Furthermore, the proposed method works for an off-policy based algorithm, which is suitable for a low-resource architecture, such as Raspberry Pi boards, for instance. It is shown that the method is able to withstand learning via exploration and still finds satisfactory convergence to significant pruning scales.

**Acknowledgements**

Dedicated to the young, naïve version of myself who fought relentlessly to achieve his dreams, and to my loving and selfless parents who fueled those dreams, as well as the peers who continued to believe in and support me.

From the bottom of my heart, I wish to thank my supervisor, Professor Wail Gueaieb. He believed in me and in my abilities even during the moments when I did not. This work would not have been possible without his continuous guidance and support. I will always remember the words he once shared with me that became a true turning point in this journey: "When a mountaineer wishes to climb a mountain, he does not take all the steps to the summit at once. He takes small increments, small steps in the right direction, and eventually reaches the summit. We must not get lost and flustered in a desperate rush to the top but focus on the hurdles directly in-front of us."

I will forever be grateful to my parents and my brother, for giving me the opportunity to see this lifelong dream come true. They have always supported me and stood in my corner for every need and every want. I am truly a blessed and privileged child to have such awe-inspiring parents. Their unwavering support has always helped me get back on my feet and continue moving forward. Without them, I would not be where I am today. I write this with nothing but gratitude and love in my heart, and I hope that I have grown into someone they are proud of.

I also wish to thank my girlfriend, and my best-friend. They carried me through one of the toughest phases of my life and continue to stand beside me with an unwavering spirit. I genuinely believe this work would not have been possible without my girlfriend's support and belief in me. Every difficult moment felt a little lighter knowing she was there. I have grown as a person through the love and care she has given me, and it has helped shape me into the individual I am becoming.

I learned many things throughout the journey of completing this thesis. I have a fun theory about it as well, the person who started this work was not the person who finished it. Which is to say, I genuinely had to grow and evolve, both mentally and practically, in order to bring this thesis to completion.

# Table of Contents

# List of Tables

# List of Figures

# Abbreviations

**ATACOM** Acting on the TAngent Space of the Constraint Manifold 2–4, 6, 16, 23, 28, 30–32, 38, 39, 41, 44, 50, 55, 57, 60, 61, 65, 71–73, 75–77, 94, 98–102, 106

**CMDP** Constrained Markov Decision Process 23, 38, 39, 73

**CSM** Constraint Space Manifold iv, 1–5, 7, 18, 31, 55–57, 71–73, 75–77, 98, 99, 106, 107

**DDPG** Deep Deterministic Policy Gradient 9–11, 31, 32, 34–38, 49, 55, 73–75, 115–117

**Deep-RL** Deep Reinforcement Learning iv, 1–5, 7–10, 13–16, 22, 25, 26, 28, 30–33, 38, 45, 72, 73, 75, 99, 106

**DOF** Degree Of Freedom 72, 92, 106

**DQN** Deep Q-Network 8, 9, 11, 32

**IMP** Iterative Magnitude Pruning iv, 48–51, 53, 62, 66, 71

**IQR** Inter-Quartile Range 80, 81

**LTH** Lottery Ticket Hypothesis iv, 1–5, 7, 26, 31, 32, 45, 50, 55, 57, 67, 71, 73–77, 88, 101, 106, 115, 117

**MDP** Markov Decision Process 8, 11, 31, 32, 38, 45, 55, 63, 65, 68, 70, 72

**OSMP** One Shot Magnitude Pruning iv, 49–51

**PPO** Proximal Policy Optimization 9–11

**ROS** Robot Operating System 107

**SAC** Soft Actor-Critic 9–11, 15, 74, 75, 115–118, 120

**TD** Temporal Difference 8, 11, 36

**TD3** Twin Delayed Deep Deterministic Policy Gradient 10, 11

**TRPO** Trust Region Policy Optimization 9–11

# List of Symbols

$\mathbf{C_{avg}}$ Average constraint violation 75, 77

$\mathbf{C_{max}}$ Maximum constraint violation 75, 77

$\mathbf{C_{\dot{q}_{max}}}$ Maximum velocity limit constraint violation 75

$\mathbf{E}$ Entropy 75

$\mathbf{J}$ Discounted reward return 75, 77, 81, 86, 88, 91, 95, 104, 105, 119–121

$\mathbf{R}$ Total reward return 75, 77

$\boldsymbol{\pi}_{\boldsymbol{\theta}}^{\boldsymbol{D}}$ Deterministic Policy with $\theta$ parameters 9, 32

$\boldsymbol{\zeta}$ Target Sparsity 52

# Chapter 1

# Introduction

The first chapter of this thesis serves as an introduction to the reader, providing a high-level understanding of the work and its context within the broader field. Section 1.1 briefly aligns the reader with the history and development of state-of-the-art works in path-planning and control tasks that employ Deep Reinforcement Learning, Constraint Space Manifold, or Lottery Ticket Hypothesis, thereby establishing the necessary background. Subsequently, Section 1.2 elaborates on the motivation and objectives driving this research. Section 1.3 summarizes the main contributions of this work, and Section 1.4 outlines the academic outputs prepared for submission to peer-reviewed venues. Finally, Section 1.5 presents an overview of the thesis structure and guides the reader through the flow of the subsequent chapters.

## 1.1 Background

This section briefly outlines important modern works and concepts that underpin the research discussed in this thesis.

**Deep Reinforcement Learning (Deep-RL).** Path-planning and control remain central to robotics, where an agent must act reliably in uncertain and dynamic environments. Deep-RL has emerged as a powerful paradigm for addressing such problems, enabling data-driven exploration of the robot's task space. Recent studies have employed Deep-RL across robotics, game playing, and continuous control tasks [15, 51, 57], leveraging deep

neural networks to approximate high-dimensional policies and value functions. The growing popularity of Deep-RL has spurred several advancements aimed at improving learning efficiency, safety, and adaptability, which are central topics in the field.

**Lottery Ticket Hypothesis (LTH).** The work by Frankle and Carbin [9] suggests that within an over-parameterized neural network, there exist smaller sub-networks, a supposed "winning ticket" that, when trained in isolation, can achieve comparable performance to the full model . Techniques such as Iterative Magnitude Pruning (IMP) and One-Shot Magnitude Pruning (OSMP) have been shown to identify these sub-networks. Applying this principle to reinforcement learning introduces the potential for more resource-efficient agents that maintain comparable control performance. Works such as [56, 54] show further exploration in applying the LTH in the Deep-RL domain.

**Constraint Space Manifolds (CSM).** In robotics and control, many tasks are defined on *constraint manifolds*, which are smooth geometric surfaces embedded within higher-dimensional configuration spaces on which the system must evolve. These constraints arise from kinematics, task geometry, contacts, and safety limits, all of which restrict feasible robot motion to a lower-dimensional subspace. Some classical works on manifold-based robotics and optimization are [49, 29], where the authors show that explicitly respecting these manifolds improves stability, accuracy, and generalization. Recent Deep-RL-based approaches extend this principle by modeling or learning constraint manifolds directly from data, enabling planners and controllers that inherently satisfy physical constraints during execution. The Acting on the TAngent Space of the Constraint Manifold (ATACOM) framework [30] provides a similar CSM structured approach to embedding physical constraints directly into an agent's learning process. It projects actions onto the tangent space of a constraint manifold, ensuring that learned behaviors remain physically feasible and safe. Incorporating this concept with Deep-RL produces policies that not only optimize rewards but also respect dynamic and geometric constraints, which is essential in robotics where physical safety and stability are *critical*.

By combining the efficiency of LTH-based subnetworks with the safety and structure provided by CSM, this work aims to produce lightweight, and reliably trainable Deep-RL agents that improve the scalability of learning complex tasks on low-resource architectures. This synthesis forms the foundation of the proposed *Trikaya* framework, which integrates these ideas into a unified and coherent methodology.

## 1.2 Motivation and Objectives

### 1.2.1 Motivation

This work is motivated by the need to overcome key limitations in current reinforcement learning methods for robotic control.

**Computational Inefficiency.** Training Deep-RL models typically requires substantial computational resources, especially as policy networks grow in depth and width. This high computational burden limits the scale of deploying such models on embedded or real-time robotic systems, where latency, power consumption, and hardware constraints impose strict limits on processing capacity. As a result, many high-performing RL agents remain confined to simulation or offline evaluation, unable to meet the real-time demands of physical robotic platforms.

**Constraint Neglect.** Conventional Deep-RL algorithms often overlook the underlying geometric and physical manifold structures inherent to robotic systems. By exploring the full action space without regard for constraints such as kinematics, contact consistency, or feasible motion subspaces, agents may engage in inefficient or unsafe exploration during training. This mismatch between the assumed and actual feasible space not only slows learning but also leads to policies that violate task-critical constraints, reducing reliability in real-world applications.

By combining the efficiency of Lottery Ticket Hypothesis with the safety constraints imposed by CSM, this research aims to develop a Deep-RL framework that is termed *Trikaya*. That enables constraint safe training of significantly pruned networks that perform on par with their fully dense counterparts, while requiring far fewer computational resources.

### 1.2.2 Objectives

The term *Trikaya* is a Sanskrit term that refers to the embodiment of three bodies, which reflects the threefold nature of the proposed approach:

- Leveraging off-policy Deep-RL algorithms that, though computationally heavier, can learn robust policies with fewer real-world interactions.
- Ensuring safe learning through CSM-based ATACOM environments.
- Employing the LTH to train significantly smaller networks that achieve near-identical performance to their dense counterparts.

Off-policy algorithms provide superior sample efficiency and faster convergence but typically employ larger networks, which poses challenges for deployment and real-time inference. Training smaller networks from scratch is often infeasible, while pruning dense models after training risks removing crucial learned representations.

The ATACOM approach [30], which leverages the tangent space of the environment's constraint manifold, enables the formation of safer and physically coherent actions during learning. This structure ensures that policy updates remain aligned with feasible directions on the manifold, improving stability and interpretability.

In a nutshell, our approach arises from the intersection of off-policy Deep-RL learning, with constraint-aligned learning on the CSM using the ATACOM environments, and pruning with re-initialization via the LTH to obtain efficient winning ticket networks (overarched by the Hypothesis 1). This idea is visually illustrated by a Venn Diagram in Fig. 1.1.

These three components work harmoniously to bridge the gap between the theoretical potential of lottery ticket networks with reinforcement learning and the practical constraints of real-world deployment. The overarching research hypothesis that guides and enables the success of this work is stated as follows:

**Hypothesis 1.** *The Lottery Ticket Hypothesis becomes more effective in identifying winning-ticket networks for reinforcement learning problems involving high-dimensional control tasks when combined with environments that enforce an action space aligned with the Null/Tangential space of the Constraint Space Manifold (such as ATACOM environments).*

Under the proposed Hypothesis 1, *Trikaya* makes the discovery of winning tickets substantially easier, enabling the training of compact and feasible networks from scratch. These networks retain fast efficient learning behavior under off-policy Deep-RL while ensuring adherence to safety and constraint requirements imposed by the CSM.

When combined with LTH, this CSM-based structured learning process enables the training of compact sub-networks that retain strong representational capacity. Traditional LTH implementations (e.g., [56]) typically achieve only 60–70% parameter reduction for high-dimensional control tasks. In contrast, *Trikaya*, through its modified mask formulation and constraint-adhering learning process, achieves reductions exceeding 90–95% while maintaining competitive task performance.

## 1.3 Contributions

The main contributions of the thesis are thoroughly highlighted in Chapter 4. They can be summarized in the following points:

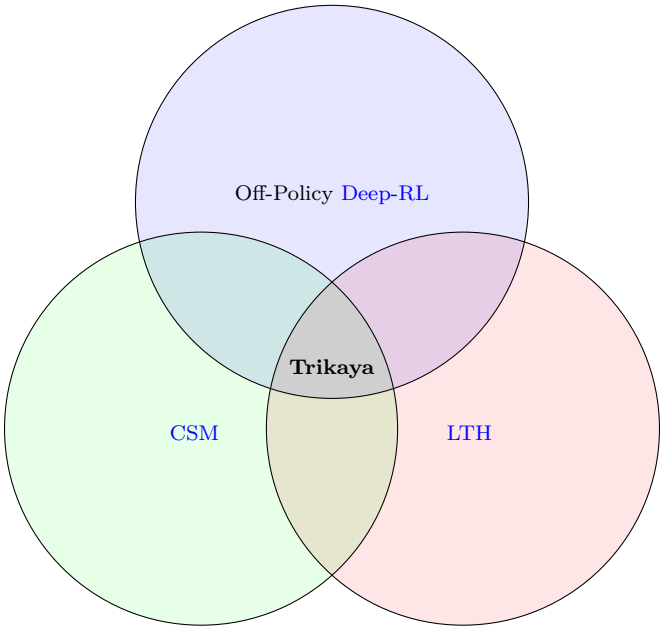



Figure 1.1: The basis of the *Trikaya* framework

- We develop the *Trikaya* framework, which enables the training of significantly pruned (90–95% smaller) networks that perform on or near par with their fully dense counterparts across multiple levels of robotic control tasks, while requiring far fewer computational resources.
- We propose a hypothesis that states the increased effectiveness of the Lottery Ticket Hypothesis in identifying winning tickets with the use of Constraint Space Manifold environments that embed their action spaces on the Constraint Manifold's tangential space.

Additionally, we present some mathematical modeling and logical formulations in Chapter 4 (Section 4.2) that successfully convey the proposed advantages of this work.

## 1.4 Scholarly Outcome

The research presented in the thesis has not been published in a peer-reviewed venue yet. However, the following article is currently in the making and should be submitted soon.

- [25]: Rushil Kumar and Wail Gueaieb. "Leveraging Constraint Space Manifolds and Lottery Tickets to Learn Complex Control Tasks with Pruned Neural Networks". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. (To be submitted). Pittsburgh, Pennsylvania, USA, Sept. 2026, pp. 1–6

## 1.5 Thesis Outline

The remainder of this thesis is organized in the following manner. Chapter 2 presents the literature review, establishing the historical and conceptual foundations of this work. Chapter 3 introduces the mathematical preliminaries underlying reinforcement learning, constraint manifolds, and the Lottery Ticket Hypothesis, and further details the methodology of the *Trikaya* framework, including pruning strategies, ticket construction, and the formulation of the unified algorithm. Chapter 4 highlights the main contributions of this work, including the central proposed hypothesis, the mathematical and logical formulations that convey the advantages of the proposed approach, and the compiled algorithms that support future reproduction of this study. Chapter 5 describes the experimental setup, simulation environments, and training configurations used to evaluate the framework, followed by the presentation and analysis of the results across multiple ATACOM environments, highlighting both the strengths and limitations of the proposed approach. Finally, Chapter 6 concludes the thesis with a discussion of the key findings, associated limitations, and potential directions for future research. Together, these chapters provide a coherent progression from theoretical formulation to experimental validation, demonstrating how *Trikaya* advances the efficiency and safety of reinforcement learning for high-dimensional robotic control by producing shortened, trainable subnetworks that retain the performance of their full-scale counterparts. This marks an important step toward improving the scalability of deep RL methods and enabling their deployment on low-resource robotic architectures.

# Chapter 2

# Literature Review

The drive to develop autonomous agents that can intelligently interact with the physical world remains a central challenge in robotics and AI [22, 23, 17, 3, 6, 12]. This pursuit seeks to move beyond purely mathematical or model-intensive decision-making pipelines, aiming instead for intelligent, data-driven applications in which agents learn to execute complex tasks within dynamic, uncertain, and unstructured environments. The core ideas underpinning this thesis draw from diverse research directions. Accordingly, this chapter examines the relevant literature across these domains, highlighting practical applications, recent advances, and the remaining gaps that motivate and justify the development of the present work.

This literature review confronts three complementary research domains that work in synergy to serve as the three required pillars of the *Trikaya* framework. In Section 2.1, we first consider the fundamental trade-offs involved in achieving efficient Deep Reinforcement Learning paradigms [50, 22]. Secondly, Section 2.2 studies works that apply the mathematical frameworks of Constraint Space Manifold, which provide a structure for embedding task rules and physical laws directly into the learning problem, thereby simplifying constrained optimization [2, 18, 21, 30]. In Section 2.3, we draw upon the Lottery Ticket Hypothesis and the broader sparse-training literature, which propose that small, efficient subnetworks within larger models can be identified and trained effectively [9, 10, 12, 45]. Finally, in Section 2.4, we provide a brief summary of the literature reviewed in this chapter and conclude by highlighting the gaps addressed through the development of the *Trikaya* framework in this research.

## 2.1 Deep Reinforcement Learning for Robotic Motion Planning

Deep Reinforcement Learning (Deep-RL) provides a general framework for sequential decision-making under uncertainty and has become a core paradigm for robotic control [50, 22]. The overarching problem motivating this line of work is how an agent can autonomously acquire behaviours through interaction, without hand-crafted controllers or accurate models. Deep extensions of RL (Deep-RL) address the related problem of operating directly on high-dimensional observations, such as images, by using neural networks as function approximators [35, 28, 14]. These capabilities are crucial for robotic motion planning and manipulation from rich sensory inputs [13, 17], but they introduce new challenges in terms of sample efficiency, safety, and network size.

### 2.1.1 Foundations: MDPs and Deep Function Approximation

The foundational problem addressed by Sutton and Barto [50] is to formalize learning from interaction as an optimization problem over trajectories. They introduce Markov Decision Process (MDP) $(S, A, P, R, \gamma)$ as the basic mathematical model and show how value functions, policies, and temporal-difference learning interlock to produce convergent control algorithms in tabular settings. Their approach provides the theoretical machinery (Bellman equations, policy evaluation and improvement, and Temporal Difference (TD) methods) that underpins most later work. What works well in this formulation is the clean separation between dynamics and decision-making, as well as the provable convergence of many algorithms in finite MDPs. The gap, for our purposes, is that these results assume discrete, low-dimensional state-action spaces and do not directly address the realities of continuous, high-dimensional robotic systems or the computational cost of function approximation.

The move to Deep-RL begins with the problem tackled by Mnih et al. [35]: how to scale value-based RL to raw visual input, where tabular methods are infeasible. Their Deep Q-Network (DQN) uses a convolutional network to approximate the action-value function $Q_\theta(s, a)$ from image observations, stabilized by two key design choices: an experience replay buffer and a slowly updated target network. The approach works remarkably well on Atari 2600 games, achieving human-level control. However, DQN remains restricted to discrete actions, operates purely in simulation, and is highly sample-hungry. The gap is therefore twofold: it is ill-suited for the continuous control required in robotics, and it does not address safety or constraint satisfaction.

Lillicrap et al. [28] target exactly this gap by asking how to extend DQN-style successes to continuous action spaces. Their Deep Deterministic Policy Gradient (DDPG) algorithm combines an off-policy actor-critic architecture with replay buffers and target networks, where a deterministic actor $\boldsymbol{\pi}_{\boldsymbol{\theta}}^{\boldsymbol{D}}(s)$ outputs continuous actions and a critic $Q_\phi(s, a)$ is trained via temporal-difference learning. The approach works well on standard continuous-control benchmarks (e.g., MuJoCo tasks), demonstrating that Deep-RL can handle continuous torques and joint velocities. In practice, however, DDPG is often brittle and sensitive to hyperparameters, and it offers no explicit mechanism for handling safety or physical constraints. The remaining gap is stability and robustness in more realistic robotic settings.

Haarnoja et al. [14] address the stability and robustness aspect by posing the problem as *maximum entropy* RL: they seek policies that maximize both expected return and entropy. Their Soft Actor-Critic (SAC) algorithm is an off-policy actor-critic method that adds an entropy term to the objective, encouraging exploration and preventing premature convergence. Empirically, SAC significantly improves sample efficiency and robustness across a range of continuous-control benchmarks. Yet, like DDPG, SAC operates in an unconstrained action space with large neural networks, and its safety properties are largely emergent. It does not reason about constraint manifolds or about whether a smaller, sparse subnetwork would suffice for the types of questions tackled in later sections of this thesis.

Schulman et al. [42] target a different failure mode of early policy-gradient methods: while policy gradients can handle continuous actions, they often suffer from unstable updates that catastrophically degrade performance when step sizes are poorly chosen. Their Trust Region Policy Optimization (TRPO) approach stabilizes learning by constraining each policy update to remain within a trust region, typically enforced via a KL-divergence bound between the old and new policies. Empirically, TRPO yields significantly more reliable learning curves on continuous-control benchmarks than unconstrained policy-gradient baselines, and it helped establish actor-critic methods as practical tools for robotics-scale control. The remaining gap is efficiency: TRPO is fundamentally on-policy, requiring fresh trajectories for each update, which leads to high sample complexity and limits direct applicability to real-world robotic training where interactions are costly.

Schulman et al. [43] address the practical complexity of TRPO by asking how to preserve trust-region-like stability while simplifying implementation. Their Proximal Policy Optimization (PPO) algorithm introduces a clipped surrogate objective (or an explicit KL penalty) that prevents overly large policy updates without requiring second-order optimization or hard KL constraints. PPO works well in practice: it is comparatively easy to tune, stable across many environments, and has become a default on-policy baseline for continuous control. However, the same fundamental limitation remains: PPO is on-policy and discards old data after each update, making it sample-inefficient for robotics. Moreover,

PPO typically relies on large dense networks and does not provide explicit mechanisms for satisfying geometric or safety constraints beyond reward shaping or external safety layers.

Fujimoto, Hoof, and Meger [11] specifically target the brittleness observed in deterministic off-policy actor-critic methods such as DDPG. They identify overestimation bias in the critic as a key source of instability and propose Twin Delayed Deep Deterministic Policy Gradient (TD3), which introduces three stabilizing modifications: clipped double Q-learning (two critics and a conservative target), delayed policy updates, and target policy smoothing. Empirically, TD3 substantially improves stability and final performance over DDPG across MuJoCo continuous-control tasks while retaining the sample-efficiency advantages of off-policy learning via replay buffers. The remaining gap is that TD3 still operates in the ambient, unconstrained action space; it inherits the same limitation as other off-policy methods in that constraint satisfaction and safety are not enforced by construction, and the learned actor and critic remain dense and computationally heavy for deployment.

To provide a more compact and unified view of the algorithms discussed, Table 2.1 and Fig. 2.1 are presented to thematically organize and summarize the core ideas, practical use cases, and trade-offs of each method. In the context of the *Trikaya* framework, these summaries serve not only as a reference, but also as a structural bridge between classical Deep-RL formulations and the constraint-aligned, sparsity-aware training paradigm developed in the subsequent sections.

#### 2.1.1.1 Policy Learning Comparisons

While the previous section discussed specific Deep-RL algorithms, it is useful to explicitly highlight the underlying on-policy versus off-policy distinction, as it directly influences efficiency in robotics.

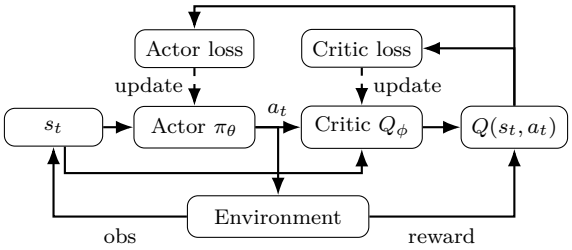
The core problem is how aggressively an RL agent can *reuse* past experience without sacrificing stability. On-policy methods such as TRPO and PPO [42, 43] address stability by updating the policy only from data generated by its current behavior. Their approach, using trust regions or clipped objectives, effectively avoids large, destructive policy updates and has become a standard choice in continuous-control benchmarks. The gap is that each policy update invalidates past data, leading to severe sample inefficiency, which is problematic for real robots [17].

Off-policy methods such as DDPG, TD3, and SAC [28, 11, 14] tackle the efficiency problem by decoupling the behavior policy from the target policy and storing past transitions in a replay buffer. Their approach enables each real interaction to be reused many

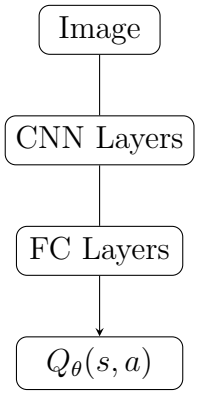
Table 2.1: High-level comparison of foundational Deep-RL algorithms.

| Algo. | Core Idea | What Works Well | Key Gap |
|---|---|---|---|
| MDP | Formalize RL using value functions, policies, and TD learning. | Clear theory; provable convergence in tabular MDPs. | Not suited for continuous, high-dimensional robotics. |
| DQN | Learn Q-values from raw images via deep CNNs. | Human-level Atari performance; stable via replay + target network. | Only discrete actions; sample-hungry; no safety handling. |
| DDPG | Deterministic actor-critic for continuous control. | Works on MuJoCo tasks; off-policy efficient. | Brittle, sensitive to tuning; no constraint satisfaction. |
| SAC | Maximize return + entropy for robust exploration. | Very stable and sample-efficient. | Unconstrained action space; large dense networks; safety is emergent. |
| TRPO | Constrain policy updates via KL trust region. | Highly stable policy improvement. | On-policy and computationally expensive; poor sample efficiency. |
| PPO | Simplify TRPO with clipping / KL-penalty. | Easy to tune; stable; widely adopted. | Still on-policy; discards old data; inefficient for robotics. |
| TD3 | Twin critics + delayed updates + smoothing. | Much more stable continuous-control learning. | Unconstrained; requires dense networks; no safety guarantees. |

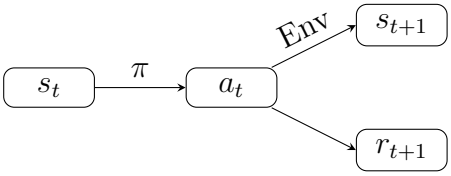
times, dramatically improving sample efficiency. What works is clear: off-policy actor-critic methods dominate in terms of data efficiency on standard benchmarks and in many robotic applications [13, 17]. The remaining gap is that this flexibility comes at the cost of more complex failure modes, including divergence due to the "deadly triad" of off-policy learning, bootstrapping, and function approximation [50]. Critically for this thesis, neither paradigm inherently addresses constraint satisfaction or network sparsity.




(a) DDPG: critic takes $s_t$ and $a_t$ as separate inputs.



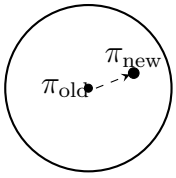

(b) DQN with convolutional encoder.




(c) MDP formulation (Sutton & Barto).

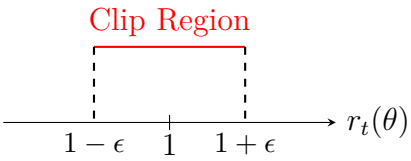
$$J(\pi) = \sum_t \mathbb{E}_{\rho_\pi}[r + \alpha\mathcal{H}(\pi(\cdot|s_t))]$$

(d) SAC: Max-entropy objective.

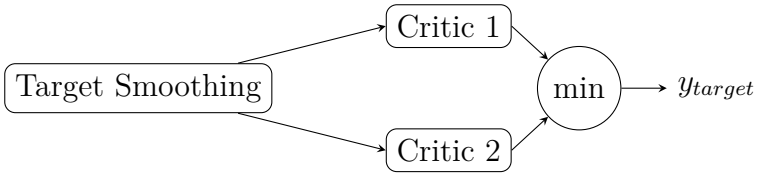



Region: $D_{KL} \leq \delta$

(e) TRPO: Update constrained inside KL-ball.




(f) PPO: Clipped probability ratio.




(g) TD3: Target uses $\min(Q_1, Q_2)$ to reduce overestimation.

Figure 2.1: Illustrative diagrams for foundational Deep-RL methods.

### 2.1.2 Few Implementations and Challenges in RL for Robotics

Beyond individual algorithms, several works classify Deep-RL methods and analyze their trade-offs for robotics.

Kober, Bagnell, and Peters [22] survey reinforcement learning in robotics with the problem of bridging the gap between RL theory and practical robotic applications. They categorize methods into policy search, value-based, and actor-critic approaches, and highlight issues such as sample complexity, safety, and prior knowledge. Their approach is primarily taxonomic, collating many empirical results to identify which methods are feasible on hardware. What works is the clear identification of policy search and actor-critic methods as more suitable for continuous robotic control. The gap, however, is that most of the surveyed methods either rely on relatively low-dimensional policy parameterizations or do not fully exploit modern deep networks; large-scale Deep-RL and its computational footprint are not yet the primary focus.

Kroemer, Niekum, and Konidaris [23] revisit this problem almost a decade later, now in the context of deep learning. They review robot learning for manipulation and emphasize the role of Deep-RL in learning complex visuomotor skills. Their approach structures the literature around representations (e.g., object-centric, end-to-end) and learning paradigms (imitation, RL, and model-based RL). They document many successes of Deep-RL and imitation learning in simulation and on real robots. What works is the demonstration that deep policies can, in principle, solve challenging manipulation tasks. Yet they also point out that these methods often require large datasets, extensive simulation, and substantial computing, and they rarely incorporate formal safety mechanisms. The gap is a lack of methods that are simultaneously sample-efficient, constraint-aware, and lightweight enough for deployment on embedded hardware.

Ibarz et al. [17] explicitly focus on the problem of how to *practically* train robots with Deep-RL. Their approach is to distill lessons from multiple real-world deployments, covering design choices such as off-policy versus on-policy learning, reward shaping, curriculum design, and safety monitoring. They show that, with careful engineering—such as the use of off-policy learning, replay buffers, and demonstration data—Deep-RL can be made more data-efficient and robust on physical systems. What works is the pragmatic recipe they provide. The gap, however, is that even with best practices, training remains expensive, heavily reliant on dense neural networks, and only weakly tied to the geometric structure of robot constraints.

Dulac-Arnold, Mankowitz, and Hester [6] broaden the problem to "real-world RL" in industrial settings, emphasizing constraints such as limited data, strict safety requirements,

and infrastructure costs. Their approach is a conceptual framework that identifies key challenges, including sample efficiency, safe exploration, non-stationarity, and system integration. This analysis serves as a checklist of obstacles that RL must overcome to be widely adopted. The gap it reveals is precisely where this thesis intervenes: most current algorithms address efficiency and robustness at the level of optimization and data reuse, rather than at the level of policy representation (e.g., constraint-aware manifolds or sparse subnetworks).

### 2.1.3 State-of-the-Art Robotic Applications Towards Efficiency and Remaining Gaps

Several works concretely demonstrate Deep-RL in robotic motion planning and control, each highlighting different aspects of efficiency and practical deployment. These works are summarized in Table 2.2 in a compact form to facilitate a concise comparative overview.

Gu et al. [13] tackle the problem of sample efficiency for visuomotor manipulation. Their approach combines off-policy actor-critic learning with normalized advantage functions, asynchronous updates, and a mix of simulated and real-world data. This enables the learning of manipulation skills, such as grasping and pushing, with significantly fewer real-world interactions than purely on-policy baselines. What works is integrating off-policy learning and replay to aggressively reuse data. The remaining gap is that constraints (e.g., joint limits and collision avoidance) are managed through environment design and low-level controllers rather than through a constraint-aware policy parameterization.

Levine et al. presents a series of works [26, 27] on end-to-end visuomotor control, where the problem is to learn manipulation policies directly from camera images to torques. Their approach uses guided policy search and large-scale data collection to train deep networks that map images to actions. These methods work well in demonstrating that rich visuomotor policies can be learned on real robots, given sufficient data and infrastructure. The gaps are the heavy data requirements, reliance on powerful computing resources, and the lack of explicit mechanisms for ensuring constraint satisfaction during exploration and deployment.

Fan et al. [8] focus on distributed multi-robot collision avoidance. The problem is safe navigation for many robots in dense environments. Their approach trains a centralized policy via Deep-RL in simulation and then deploys it in a decentralized manner, where each robot relies on local sensing. The results show robust collision avoidance and emergent cooperative behaviours. What works is Deep-RL's ability to capture complex interaction

dynamics between agents. The gap, again, is that safety is enforced empirically and statistically; there is no guarantee that trajectories remain within a well-defined constraint manifold.

Rudin et al. [41] address the efficiency problem for legged locomotion by exploiting massive parallel simulation. They train high-performance quadrupedal locomotion policies in minutes using SAC and thousands of parallel environments. This approach impressively reduces wall-clock training time. However, the efficiency gain comes primarily from hardware parallelism rather than from algorithmic or representational improvements: the networks remain large, and constraint handling (e.g., contacts and joint limits) is handled by the physics engine and low-level controllers rather than by the policy structure itself.

Xiao et al. [55] examines coverage path planning and navigation with sparse rewards using Deep-RL, combining hindsight experience replay and multimodal fusion. Their method addresses the problem of efficient exploration in partially known environments. The approach improves credit assignment and enables effective learning under sparse feedback. The gap is that the resulting policy still reasons over the full, unconstrained action space and relies on dense networks; efficiency is improved at the algorithmic level rather than through structural constraints or sparsity.

Taken together, these works demonstrate that Deep-RL can solve challenging robotic motion planning and control problems, particularly when combined with off-policy data reuse and large-scale simulation. They also leave two central issues largely open and motivate the directions pursued in this thesis:

1. Embed *constraint satisfaction* into the policy representation itself, rather than relying on penalties or external safety filters; and

2. Reduce the *computational footprint* of Deep-RL policies by exploiting the existence of smaller, more efficient subnetworks.

Dulac-Arnold, Mankowitz, and Hester [6] and Brunke et al. [3] both conclude that simply choosing between on-policy and off-policy methods is insufficient for real-world robotics. Their analyses propose hybrid or structured approaches, such as constrained MDPs, model-based components, and safety layers that sit atop standard algorithms. What works is a clearer understanding of the design space.

The gap they explicitly leave is at the *representation level* for designing policies whose action spaces and network structures inherently encode safety and efficiency. The *Trikaya* framework responds to this gap by combining:

- off-policy Deep-RL (for data efficiency),
- constraint-space manifold parameterizations (for geometric safety), and
- lottery-ticket-style sparse subnetworks (for computational efficiency).

The following section reviews geometric approaches based on constraint-space manifolds, such as ATACOM, which address the first point by enforcing feasibility at the action-parameterization level. The subsequent section then turns to network sparsity and the Lottery Ticket Hypothesis, which address the second point by arguing that significant Deep-RL policies may contain much smaller "winning ticket" subnetworks that retain performance while reducing computational cost.

## 2.2 Geometric Approaches for Constrained Motion Planning

Robotic systems operating in real-world environments are almost invariably subject to constraints arising from physical laws, hardware limitations, and task-specific requirements. These constraints define a feasible subset of the robot's configuration and control space that is typically of much lower dimension than the ambient space itself. As a result, the central problem motivating this line of work is not merely how to plan or learn effective motions, but how to do so *while remaining on a feasible set of measure zero embedded within a high-dimensional space*. Classical motion planning and control frameworks have addressed this challenge through explicit constraint handling, projection methods, and optimization-based formulations [44, 32, 23]. However, as task complexity and system dimensionality increase, these approaches often struggle to simultaneously provide computational efficiency, scalability, and formal guarantees of feasibility.

This section reviews geometric approaches that formalize constraints as smooth manifolds embedded in the robot's configuration space. The discussion follows a problem-approach-gap perspective, moving from foundational formulations of constraint manifolds, to projection- and learning-based planning algorithms, and finally to recent efforts that integrate geometric feasibility directly into reinforcement learning through tangent-space action parameterizations, such as ATACOM.

Table 2.2: Representative robotic RL applications that emphasize efficiency, and the remaining limitations that motivate this thesis.

| **Approach** | **Problem Focus** | **What Works** | **Key Gap** |
|---|---|---|---|
| Off-policy visuomotor RL with replay (real+sim data reuse) [13] | Sample-efficient visuomotor manipulation. | Off-policy actor–critic style learning and replay enable aggressive reuse of real and simulated data. | Constraints are typically handled indirectly (environment design, controllers), not embedded in the policy representation. |
| Large-scale end-to-end visuomotor control (images→torques; guided policy search) [26, 27] | End-to-end visuomotor control at scale. | Demonstrates that rich visuomotor policies can be learned on real robots with sufficient data and infrastructure. | High data and compute requirements, and constraint satisfaction is not explicitly guaranteed during learning or deployment. |
| Centralized training, decentralized execution for multi-robot navigation [8] | Distributed multi-robot collision avoidance in dense settings. | RL captures complex interaction dynamics and yields robust decentralized behaviors from centralized training. | Safety remains empirical (statistical success), not guaranteed through feasibility-by-construction parameterizations. |
| Massively parallel simulation for fast locomotion learning (SAC) [41] | Reduce wall-clock training time for legged locomotion. | Parallelism drastically reduces training time, enabling rapid iteration and high-performance policies. | Efficiency is hardware-driven, networks remain large, and constraints are mostly handled by physics and low-level control. |
| Sparse-reward navigation with replay augmentation (HER) and multimodal fusion [55] | Coverage path planning and navigation under sparse feedback. | Improves exploration and credit assignment under sparse rewards. | Policies still operate over unconstrained spaces and remain dense, efficiency is improved algorithmically rather than structurally. |

### 2.2.1 Foundations: Constraint Manifolds and Tangent-Space Reasoning

To systematically address constraints in robot motion planning, it is common to formulate the problem in the robot's *Configuration Space* (C-space), where each point $q \in Q$ represents a complete specification of the robot's joint variables. A set of $k$ independent equality constraints can be described by a smooth function $F(q) = 0$, defining a CSM

$$\mathcal{M}_c = \{q \in Q \mid F(q) = 0\},$$

which forms a smooth, low-dimensional surface embedded in the higher-dimensional ambient space [2, 18].

The key geometric objects required to reason on this manifold arise from differential geometry. At any point $q \in \mathcal{M}_c$, the *tangent space* $T_q\mathcal{M}_c$ characterizes all instantaneous velocity directions that satisfy the constraints. This space is given by the null space of the *constraint Jacobian* $J_c(q) = \partial F/\partial q$, such that any feasible velocity $\dot{q}$ must satisfy $J_c(q)\dot{q} = 0$. Projection operators constructed from $J_c$ and its pseudoinverse allow arbitrary points or directions in the ambient C-space to be mapped back onto the manifold, thereby correcting constraint violations.

This geometric formulation provides a powerful conceptual framework: rather than viewing constraints as penalties or inequalities to be enforced during optimization, feasibility is represented as a structural property of the space in which planning and control occur. What works particularly well in this setting is the ability to decouple the representation of task feasibility from the objective of performance optimization, allowing algorithms to operate locally within the tangent space of valid motions. The remaining gap, however, is that this framework alone does not prescribe how to efficiently explore or learn policies on the manifold in high-dimensional systems, particularly when constraints are complex or partially unknown.

### 2.2.2 Few Foundational Applications in Constraint Manifolds

One of the earliest algorithmic responses to the manifold formulation of constraints is the adaptation of sampling-based motion planners to operate on embedded feasible sets. The central problem addressed by these methods is how to explore $\mathcal{M}_c$ (the manifold) efficiently when random samples drawn from the ambient configuration space $\mathcal{Q}$ (the C-space) almost surely fail to satisfy the constraints due to the negligible volume of the manifold relative to $\mathcal{Q}$.

Stilman [46] and Hauser [16] establish early formulations of constrained planning for manipulation and contact-rich tasks, where feasibility is maintained by explicitly enforcing constraint satisfaction during tree expansion. Their work highlights that constraint-consistent sampling and local projection are essential for maintaining connectivity in narrow, low-dimensional feasible regions embedded in high-dimensional spaces.

Building on this foundation, Berenson et al. [2] and Jaillet and Porta [18] propose projection-based variants of Rapidly-exploring Random Trees (RRTs) for constrained planning. Their approach samples points in the ambient space and then projects promising candidates onto the manifold using numerical solvers derived from the constraint Jacobian. The resulting planners, such as the Constrained Bi-directional RRT (CBiRRT), construct trees that are guaranteed to lie on $\mathcal{M}_c$, enabling the solution of complex manipulation tasks involving closed-chain kinematics, contact constraints, and articulated object interaction.

Kingston, Moll, and Kavraki [21] generalize this perspective by providing a unifying theoretical treatment of sampling-based planning on manifolds. They formalize conditions under which probabilistic completeness and asymptotic optimality can be preserved when planners are restricted to implicitly defined constraint manifolds. What works particularly well in this framework is the rigorous connection it establishes between differential-geometric properties of $\mathcal{M}_c$ and the convergence behavior of sampling-based algorithms.

Complementary approaches extend projection-based planning to trajectory optimization and hybrid methods. For example, Toussaint [52] integrate logical and geometric constraints into continuous optimization-based motion planning, while Ratliff et al. [38] and Kalakrishnan et al. [19] demonstrate how constraints can be incorporated into trajectory optimization frameworks through cost shaping and feasibility projection. These methods offer smoother, locally optimal solutions once a feasible region has been identified, and they are often used in conjunction with sampling-based planners to refine constraint-consistent paths.

What works well across this family of methods is their generality and theoretical grounding. They leverage the well-understood exploration properties of sampling-based planners and the convergence properties of optimization-based refinements, while enforcing feasibility through explicit geometric projection or constraint-consistent updates. This combination enables the solution of a wide range of high-dimensional, contact-rich, and closed-chain planning problems that are otherwise intractable for unconstrained planners.

The gap becomes apparent, however, in tightly constrained or high-dimensional systems. Projection operations can be computationally expensive and numerically fragile, particularly when the constraint Jacobian is ill-conditioned or when the manifold contains narrow passages. Moreover, these methods remain largely disconnected from learning-

based control paradigms: they excel at computing feasible paths offline, but they do not address how a policy can be learned to act directly on the manifold through interaction, nor how such policies can be represented efficiently for real-time deployment.

### 2.2.3 Projection-Based Planning on Constraint Manifolds

We now narrow this discussion to projection-driven planners that explicitly enforce feasibility during exploration, and to variants that improve robustness by proposing motions in local tangent parameterizations near $\mathcal{M}_c$, which is the manifold.

Berenson et al. [2] and Jaillet and Porta [18] propose projection-based variants of Rapidly-exploring Random Trees (RRTs) for constrained planning. Their approach samples points in the ambient space and then projects promising candidates onto the manifold using numerical solvers driven by the constraint Jacobian, producing planners, such as CBiRRT and rapidly-exploring manifold constructions. What works well in this family of methods is their generality: they leverage the exploration properties of sampling-based planners while enforcing feasibility through geometric projection, enabling complex manipulation tasks that include closed-chain kinematics and contact constraints.

A complementary line of work targets the numerical fragility and repeated cost of explicit projection by reformulating how local exploration is performed near $\mathcal{M}_c$. Kim, Park, and Suh [20] address the problem of maintaining feasibility during tree expansion by proposing Tangent Bundle RRT (TB-RRT), which constructs motion proposals in a local tangent parameterization and uses feasibility corrections to remain close to the constraint set. What works is improved robustness in tightly constrained problems where naive project-after-sample can repeatedly fail or collapse to boundary behavior. The gap, however, is that these methods remain planners: they deliver feasible paths, but do not directly yield a closed-loop policy that can react online through interaction.

Several works further emphasize that constrained planning pipelines are typically modular, relying on a planner for feasibility and a separate controller for execution. Stilman [47] address task-constrained manipulation by planning directly in joint space under task constraints, showing that explicit constraint handling can be integrated into manipulation planning at scale. Şucan and Chitta [48] tackle the practical problem of enforcing constraints inside general-purpose planning frameworks by approximating constrained configuration spaces to make planning computationally tractable. More recently, toolchains such as Mirabel et al. [33] (and follow-up constrained manipulation planning work [34]) focus on providing stable, reusable software abstractions for planning under implicit constraints, which is critical for deploying manifold-based methods in larger robotics systems.

Taken together, projection-based and tangent-parameterized planners establish a strong feasibility-first foundation: constraints are enforced geometrically rather than via penalties, and complex constrained tasks become solvable in practice. The remaining gap, and the motivation for the learning-based direction reviewed next, is that these methods are still largely disconnected from reinforcement learning: they do not explain how an agent can *learn* to act on $\mathcal{M}_c$ directly from interaction, nor how to represent such policies efficiently for training and deployment [21].

### 2.2.4 Learning Constraint Manifolds from Data

In many realistic robotic settings, the constraint function $F(q)$ is not analytically available or is too complex to model explicitly. This motivates a second line of work that treats the constraint manifold itself as an object to be learned from data.

Rayas Fernández et al. [39] address the problem of unknown or implicit constraints by learning manifold representations directly from data. Their approach studies two families of models: (i) latent-variable manifold representations (e.g., VAEs) and (ii) neural implicit constraint representations, including the Equality Constraint Manifold Neural Network (ECoMaNN). What works well is the ability to generalize beyond a finite set of samples or demonstrations, enabling constrained planning even when explicit analytic constraints are unavailable. The gap, however, is that generalization must be reliable across the state space: small geometric errors can produce feasibility drift, and many pipelines still require explicit projection or search procedures that are computationally heavy.

A related direction focuses on improving constrained sampling and exploration when feasibility is learned rather than specified. Razmjoo et al. [40] tackle the practical issue that sample-then-project strategies often waste computation and accumulate samples near constraint boundaries. Their approach uses a product-of-experts formulation to combine optimality and feasibility distributions, enabling a project-then-sample strategy that empirically improves exploration for tasks that include manifold constraints. What works is the observed improvement in sampling efficiency and diversity under constraints. The remaining gap is that such methods still primarily strengthen the planning layer; they do not, by themselves, provide a reinforcement learning policy that learns closed-loop behaviors with guaranteed constraint satisfaction.

Finally, recent work begins to incorporate additional structure (e.g., physics priors) to reduce data demands and stabilize learned constraint representations. For example, physics-informed constrained planning formulations aim to make manifold learning more data-efficient and less brittle under distribution shift [36]. What works is stronger inductive

bias when demonstrations are limited or noisy. The open question that persists is how to translate learned manifold structure into *policy learning* that is both safe by construction and computationally efficient, particularly when the downstream learner is a Deep-RL agent whose actor/critic networks are typically dense. Some These works are summarized in Table 2.3 in a compact form to facilitate a concise comparative overview.

Table 2.3: Safe RL mechanisms versus manifold-embedded action parameterizations for constraint satisfaction.

| **Approach** | **Problem Focus** | **What Works** | **Key Gap** |
|---|---|---|---|
| CMDP-constrained policy optimization (trust-region style safety) [1] | Safe RL with explicit cost constraints during policy improvement. | Principled updates that enforce cost constraints (in expectation) while improving reward. | Safety is handled through optimization constraints, feasibility is not guaranteed by construction at the representation level. |
| Action "safety layer" via local projection to a safe set [5] | Reduce unsafe actions during training and deployment with minimal changes to the base RL algorithm. | Practical reduction in unsafe actions through action correction before execution. | Safety is an external correction layer, the underlying policy still proposes unconstrained actions. |
| Tangent-space (manifold-embedded) action parameterization (ATACOM) [30] | Enforce feasibility by restricting actions to the constraint manifold's tangent space. | Feasibility is enforced geometrically by design, allowing RL to optimize within admissible motions. | Actor and critic are typically dense, compute footprint remains large for deployment on constrained hardware. |
| Manifold-embedded safe RL at scale (higher-dimensional systems) [31] | Demonstrate scalability of manifold-embedded formulations while preserving feasibility behavior. | Empirically retains strong feasibility properties in higher-dimensional tasks. | Efficiency at the network representation level is still not the focus, dense policies remain the default. |
| Safe RL survey and design-space framing [3] | Synthesize safe RL mechanisms and assumptions for real systems. | Clarifies the landscape of safety objectives, constraints, and practical failure modes. | Representation-level solutions that unify feasibility and efficiency remain limited, especially for robotics. |

### 2.2.5 Manifold-Embedded Reinforcement Learning and Safe Exploration

A distinct shift in perspective is introduced by approaches that embed the geometric structure of the constraint manifold directly into the reinforcement learning formulation. The core problem addressed here is the inherent risk of constraint violation during exploration: standard RL algorithms operate in an unconstrained action space and rely on penalties or external safety layers to discourage infeasible behavior, which can lead to unstable learning or overly conservative policies [3].

A broad class of safe RL methods address this problem at the Constrained Markov Decision Process (CMDP) level by enforcing constraints in expectation. Achiam et al. [1] propose Constrained Policy Optimization (CPO), where policy updates are restricted to satisfy cost constraints while improving reward, producing principled trust-region style updates under safety limits. In parallel, "safety layer" style methods such as Dalal et al. [5] address the same problem by projecting a proposed action onto a locally safe set before execution. What works in these approaches is that they provide practical mechanisms to reduce unsafe actions during training. The gap is that safety is typically handled as an *auxiliary optimization objective* (cost constraints, penalties, or projections) rather than as a geometric property of the policy parameterization; feasibility is encouraged or corrected, but not guaranteed by construction.

The ATACOM framework proposed by Liu et al. [30] addresses this problem by restricting the agent's action space to the tangent space of the constraint manifold. Rather than outputting raw control commands, the policy produces tangent-space coordinates that are mapped through a null-space basis of the constraint Jacobian to generate feasible accelerations or velocities. Since tangent-space actions satisfy the instantaneous constraint conditions by construction, feasibility is enforced geometrically rather than through reward shaping. Algorithm 2.1 shows the original ATACOM environment algorithm [30]. Follow-up work extends this direction and empirically demonstrates that manifold-embedded formulations can scale to higher-dimensional systems while retaining strong feasibility behavior [31].

What works particularly well in this formulation is the separation between safety and performance. Constraint satisfaction becomes a property of the action parameterization, allowing the RL algorithm to focus on maximizing task reward within the admissible motion space. The remaining gap, and the point of departure for this thesis, is that manifold-embedded RL policies are typically realized using large, dense actor and critic networks. While feasibility can be enforced by design, the computational footprint of the resulting models remains substantial, limiting practicality on resource-constrained platforms. This

directly motivates integrating manifold-embedded RL with network sparsity and lottery-ticket-style subnetwork discovery.

**Algorithm 2.1** ATACOM environment algorithm [30]

**Input:** **Constraint:** $f$, $g$, $J_f$, $J_g$, $b_f$, $b_g$. **Scale params:** $K_c$, $K_f$, $K_g$. **Time step:** $\Delta T$.

**for** each episode **do**
  Initial feasible state $s_0$, slack variable $\mu_0$.
  **for** each time step $k$ **do**
    Sample policy action $\alpha_k \sim \pi(\cdot \mid s_k)$.
    Observe $q_k$, $\dot{q}_k$ from $s_k$.
    Compute $J_{c,k} = J_c(q_k, \mu_k)$, $\psi_k = \psi(q_k, \dot{q}_k)$, $c_k = c(q_k, \dot{q}_k, \mu_k)$.
    Compute the RCEF nullspace/tangent basis $N_c^R$.
    Compute tangent–space acceleration: $\begin{bmatrix} \ddot{q}_k \\ \dot{\mu}_k \end{bmatrix} \leftarrow - J_{c,k}^{\dagger} \Big[ K_c \, c_k + \psi_k \Big] + N_c^R \, \alpha_k$
    Clip joint acceleration $\ddot{q}_k \leftarrow \text{clip}(\ddot{q}_k, a_l, a_u)$.
    Integrate slack: $\mu_{k+1} = \mu_k + \dot{\mu}_k \, \Delta T$.
    Apply control $a_k = \Lambda(\ddot{q}_k)$ to the environment.
    Observe next state $s_{k+1}$ and reward $r_k$.
    Provide transition tuple $(s_k, a_k, s_{k+1}, r_k)$ to the RL algorithm.

## 2.3 Network Sparsity and the Lottery Ticket Hypothesis

The computational demands of Deep Reinforcement Learning extend beyond data collection and simulation infrastructure to the neural network architectures used to represent policies and value functions. In robotics, where on-board computation, memory, and power are limited, the reliance on large, densely connected networks presents a significant barrier to deployment. The central problem motivating this line of work is therefore how to retain the expressive and optimization benefits of over-parameterized models while reducing their computational footprint to a level compatible with real-world robotic systems.

This section reviews the literature on network sparsity and the Lottery Ticket Hypothesis (LTH), with particular emphasis on how these ideas transfer to the non-stationary, sequential decision-making setting of reinforcement learning. The discussion again follows

a problem-approach-gap structure, moving from foundational observations about over-parameterization, to the discovery of sparse "winning tickets," and finally to recent work on sparse and dynamic training in Deep-RL.

To provide a compact view of how these ideas relate to the broader safety and deployment discussion in this chapter, Table 2.4 summarizes the key threads at the representation level. This comparison makes the central gap explicit despite strong progress on both safety and efficiency, sparsity is still typically studied independently of geometric constraint parameterizations, motivating the integrated direction favoured in this thesis.

### 2.3.1 Over-Parameterization and the Limits of Sparse-From-Scratch Training

Modern deep learning practice has converged on the use of highly over-parameterized neural networks, not because the underlying tasks necessarily require such capacity, but because large models tend to exhibit favorable optimization properties, including smoother loss landscapes and more reliable gradient flow. In supervised learning, this phenomenon has been linked to improved convergence and generalization, and similar trends are observed in Deep-RL, where large actor and critic networks often stabilize training in complex environments.

An intuitive alternative is to train sparse networks directly, thereby reducing computational cost from the outset. However, empirical studies consistently show that networks initialized with random sparse connectivity patterns underperform compared to dense networks that are subsequently pruned to the same sparsity level [9, 7]. The primary explanation offered in the literature is that random sparsity disrupts information flow and weakens gradient signals during early training, preventing the network from discovering effective internal representations.

What works in this analysis is the recognition that *structure and initialization* play a critical role in trainability, not merely the number of parameters. The gap that follows is how to identify sparse structures that retain the favourable optimization dynamics of dense models, particularly in the presence of the non-stationary data distributions and bootstrapping effects that characterize reinforcement learning.

### 2.3.2 The Lottery Ticket Hypothesis and Rewinding

The Lottery Ticket Hypothesis, introduced by Frankle and Carbin [9], provides a concrete formulation of this problem. The hypothesis proposes that within a randomly initialized

dense network, there exists a smaller subnetwork termed as a “winning ticket", which, when trained from its original initial weights, can achieve performance comparable to the full model. The defining features of a winning ticket are therefore not only its sparse connectivity pattern, but also the specific initial values of its surviving weights.

Follow-up work by Frankle et al. [10] demonstrates that, for larger architectures and more complex datasets, resetting the subnetwork to an early training checkpoint rather than to iteration zero, a procedure known as *rewinding*, significantly improves stability and final performance. This result strengthens the central claim of the LTH by showing that favourable training trajectories can be preserved even as the network is progressively sparsified.

What works particularly well in this body of work is the empirical demonstration that high-performance models need not be dense and that sparse subnetworks can be both trainable and competitive when appropriately initialized. The gap, however, lies in transferring these insights to reinforcement learning, where the training distribution evolves as the policy changes and where performance depends not only on function approximation accuracy but also on exploration dynamics and long-horizon credit assignment.

### 2.3.3 Methodology for Discovering and Validating Winning Tickets

The standard methodology for identifying winning tickets follows an iterative three-stage process [9]. First, a dense network is trained to converge on the target task. Second, a fraction of the weights with the smallest magnitudes is pruned, defining the sparse architecture of a candidate subnetwork. Third, the remaining weights are *reset* to their original initialization values or to an early checkpoint in the rewinding variant.

The critical observation emerging from this procedure is that the sparse subnetwork with reset weights trains effectively, while an identically structured network initialized with new random weights typically fails to reach comparable performance. This isolates the importance of the initial conditions and supports the interpretation of winning tickets as particularly favourable combinations of structure and initialization.

What works in this methodology is its conceptual clarity and reproducibility across a wide range of supervised learning benchmarks. The gap, from the perspective of Deep-RL, is that the procedure is computationally expensive and was originally designed for static datasets. Its direct application to reinforcement learning, where data is generated online and the policy evolves continuously, raises questions about when and how pruning and resetting should be performed without destabilizing learning.

### 2.3.4 Sparse and Dynamic Training in Deep Reinforcement Learning

Recent work has begun to explicitly examine how sparsity and subnetwork discovery translate to the reinforcement learning setting. Graesser et al. [12] provide a systematic evaluation of sparse training methods across a range of RL benchmarks, with the central problem being whether aggressive parameter reduction can be achieved *without* destabilizing the coupled actor-critic optimization. Their approach compares pruning-at-initialization, gradual magnitude pruning, and sparse-from-scratch baselines under common RL algorithms. What works is that meaningful sparsity is often achievable while retaining competitive returns. The gap they surface, however, is that RL is uniquely sensitive: bootstrapping targets, replay-induced non-stationarity, and exploration noise can amplify the optimization fragility that sparsification introduces.

A complementary question is whether *lottery-ticket-like* effects exist in RL beyond raw return matching. Vischer, Lange, and Sprekeler [53] study the problem of identifying minimal task representations in deep RL through the lens of pruning and subnetwork structure. Their approach probes whether small subnetworks can preserve task-relevant representations (and how these representations vary across tasks), rather than focusing solely on final score. What works is the evidence that pruning can reveal compact representations that still support strong policies on a given task. The remaining gap is robustness and transfer: the discovered subnetworks can be task- and seed-dependent, and the analysis does not directly translate into a generally reliable recipe for producing sparse policies across diverse robotic environments.

More recently, Obando-Ceron et al. [37] revisit lottery-ticket-style questions in deep RL through a continual-learning perspective, addressing the problem that RL agents are often expected to adapt across changing task distributions. Their approach investigates how sparsity patterns and rewinding-like procedures interact with non-stationary learning dynamics. What works is that this framing makes explicit an important reality for robotics: even if a sparse subnetwork performs well *once*, it may not remain a stable substrate under distribution shift. The gap, for our purposes, is that these insights are largely algorithmic and diagnostic; they do not yet provide an integrated mechanism for combining sparsity with *geometric* feasibility constraints, where the action parameterization itself is already structured.

Dynamic sparse training methods, inspired by RigL-style connectivity evolution [7], extend the sparse-training story by allowing the network topology to change during training. The core problem here is that a *fixed* sparse mask may block gradient flow early, when RL

signals are already weak and noisy. RigL-like updates mitigate this by periodically pruning low-utility connections while regrowing new ones using gradient information, effectively tracking which pathways matter as learning progresses. This idea has been adapted to RL directly: Sokar et al. [45] demonstrate that dynamic sparsity can maintain, and in some settings improve, policy performance while substantially reducing parameter counts and compute. What makes these approaches work is their ability to co-evolve connectivity with the agent's learning phase, rather than committing to a potentially poor sparse structure at initialization.

The remaining gap is twofold. First, sparse and dynamic-sparse RL can require careful tuning (update frequency, regrowth rules, sparsity schedules) to avoid destabilizing learning, limiting out-of-the-box robustness. Second, and more fundamentally for this thesis, most sparse RL methods treat sparsity as an architecture-level optimization that is largely *decoupled* from task geometry. They typically operate in unconstrained action spaces and do not account for constraint-aware policy parameterizations, such as tangent-space actions on a constraint manifold. As a result, the interaction between (i) sparsity-induced representational bottlenecks and (ii) manifold-embedded feasibility guarantees remains largely unexplored.

### 2.3.5 Bridge to the *Trikaya* Framework

Taken together, the literature on geometric constraint manifolds and network sparsity reveals complementary strengths and limitations. Manifold-based approaches, exemplified by ATACOM, provide strong feasibility and safety guarantees by embedding constraints directly into the action representation. Sparse and lottery-ticket-based methods, in contrast, offer a principled route toward reducing the computational footprint of deep models without sacrificing performance, and recent analyses suggest that sparsity in RL is feasible but uniquely sensitive to instability and non-stationarity [12, 37].

The central gap that emerges is the lack of a unified framework that addresses both dimensions simultaneously. Existing work largely treats geometric feasibility and computational efficiency as separate design objectives. This thesis positions the *Trikaya* framework as a concrete integration of these ideas: off-policy Deep-RL algorithms operating on constraint manifolds through tangent-space parameterizations, realized using lottery-ticket-style sparse subnetworks. The core hypothesis is that winning tickets can be discovered *within* the manifold-embedded formulation itself, yielding policies that are not only safe by construction, but also efficient enough for practical deployment on real robotic systems.

Table 2.4: Network sparsity and lottery-ticket style subnetwork discovery, with emphasis on RL-specific challenges.

| **Approach** | **Problem Focus** | **What Works** | **Key Gap** |
|---|---|---|---|
| Rewinding for winning-ticket stability [10] | Stabilize lottery-ticket subnetworks by resetting to early checkpoints. | Rewinding improves stability and performance for larger models and harder tasks. | Still expensive, and RL introduces additional instability beyond supervised training. |
| Dynamic sparse training with gradient-based regrowth (RigL-style) [7] | Avoid sparse-from-scratch failures by evolving connectivity during training. | Topology evolution restores gradient flow by pruning and regrowing connections using gradient information. | Requires tuning of update frequency and schedules, and interaction with RL dynamics can be delicate. |
| Benchmarking sparsity methods in deep RL [12] | Test whether aggressive parameter reduction is possible without destabilizing actor–critic learning. | Meaningful sparsity is often achievable without large return loss, and failure modes are characterized. | RL is uniquely sensitive (bootstrapping, replay non-stationarity, exploration noise), stability can degrade fast. |
| Pruning to reveal compact RL representations (lottery-ticket effects in RL) [53] | Identify minimal task representations via pruning, beyond raw return matching. | Evidence that pruning can expose compact representations that still support strong policies on a task. | Subnetworks can be task- and seed-dependent, robustness and transfer remain open. |
| Lottery tickets under non-stationarity (continual RL perspective) [37] | Study how sparsity patterns and rewinding interact with distribution shift and continual adaptation. | Makes explicit how non-stationarity affects sparsity stability and the reliability of rewinding-like procedures. | Insights are largely diagnostic, and do not yet unify sparsity with geometric feasibility constraints. |
| Dynamic sparsity in RL (mask evolution during policy learning) [45] | Maintain performance with fewer parameters by co-evolving masks during RL training. | Dynamic masks can preserve and sometimes improve returns while reducing compute and parameter counts. | Often needs careful tuning, and sparsity is still treated as decoupled from task geometry and constraint manifolds. |

## 2.4 Summary

This literature review has established three critical research pillars relevant to advancing the Framework *Trikaya*. Deep Reinforcement Learning provides a robust framework for learning complex behaviours from high-dimensional inputs, though challenges remain in

sample efficiency, stability, and constraint handling [6, 17]. Geometric approaches through Constraint Space Manifolds offer a principled method for ensuring safety and constraint satisfaction by construction, culminating in ATACOM-style formulations that embed constraints into the action space itself [2, 18, 30]. Finally, the Lottery Ticket Hypothesis and sparse-training literature present a compelling direction for improving computational efficiency through network sparsity, with emerging evidence that highly sparse policies can retain strong performance [9, 12, 45].

The integration of these domains forms the foundation for developing efficient, safe, and practical autonomous robotic systems capable of operating in complex real-world environments. In the remainder of this thesis, *Trikaya* is created as a concrete instantiation of this integration: off-policy Deep-RL algorithms acting on constraint manifolds through ATACOM-style tangent-space parameterizations, realized with lottery-ticket-style sparse networks that preserve safety while reducing computational footprint.

# Chapter 3

# Background and Preliminaries

This chapter outlines the mathematical background underlying the approach used by *Trikaya*. Deep-RL agents generally operate within continuous or discrete-state MDPs, deploying actor and critic networks as neural function approximators. Establishing this foundational groundwork is essential before examining the hybrid, constrained MDPs employed in the remainder of the thesis.

We begin by reviewing the mathematical preliminaries of deterministic policies and Bellman equations, which define the learning objective of a DDPG agent trained on a dense network. This serves as the baseline learning paradigm against which subsequent methods are evaluated. The analysis is then extended to constrained MDPs through the ATACOM framework, which introduces explicit geometric and dynamic constraints that project the state–action space onto the tangent space of the constraint manifold. In this formulation, feasible actions are determined by projection operators that inherently restrict policy updates to the admissible tangential subspace of the control manifold.

Building on this constrained formulation, the next section introduces the concept of lottery ticket networks, obtained by applying the LTH to the actor and, optionally, to the critic. The resulting subnetworks define a lower-dimensional optimization landscape, whose theoretical effects on gradients, rewards, and sample efficiency are examined. These derivations connect the core research objective to the geometry of the constrained space and demonstrate how reduced network dimensionality can preserve and, in some cases, enhance the learning efficacy within these environments.

The progression of this chapter follows the conceptual flow:

$$\text{Dense DDPG} \;\rightarrow\; \text{DDPG on CSM (via ATACOM)} \;\rightarrow\; \text{Lottery Tickets on CSM}.$$

This forms the basis for the subsequent subsections on training methodology, pruning strategies, and the construction of the proposed ticket networks.

The chapter is structured as follows. Section 3.1 introduces the essential mathematical formulation of this work, followed by Section 3.2, which discusses the foundations of ATACOM and reinforcement learning. Section 3.3 and Section 4.2 develop the theoretical motivation for applying the LTH within constrained environments, explaining why subnetworks benefit from action spaces embedded in tangential manifolds. Section 3.4 reviews the mathematical basis of the pruning procedures.

## 3.1 Reinforcement Learning

This section introduces the mathematical preliminaries underlying the Deep-RL algorithms used in this work. DDPG represents a significant advancement for continuous action-space problems, where traditional DQNs are ineffective. In particular, DDPG addresses the challenge of optimizing value functions of the form.

$$\max_{a \in \mathcal{A}} Q(s, a),$$

where $a$ denotes the action selected in state $s$ from the continuous action set $\mathcal{A}$, and $Q$ is the action-value function formally defined later in (3.3). Here, the action space satisfies $\mathcal{A} \subseteq \mathbb{R}^m$, with $m$ representing the dimensionality of the task.

DDPG provides a practical solution by learning a deterministic policy $\boldsymbol{\pi}_{\boldsymbol{\theta}}^{\boldsymbol{D}}(s) : \mathcal{S} \to \mathcal{A}$ that directly outputs the action maximizing the Q-function. This removes the need for explicit maximization over a continuous action space during execution and enables efficient learning in high-dimensional continuous control tasks. A high-level training architecture diagram is given in Fig. 3.1 for a quick overview of the algorithm.

### 3.1.1 Markov Decision Process

Before delving into the specifics of DDPG, we first establish the general mathematical framework of MDPs that underlies reinforcement learning algorithms.

**Definition 3.1** (Markov Decision Process)**.** *A Markov Decision Process (MDP) is defined by the tuple* $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$*, where:*

- $\mathcal{S}$ *is the state space,*

- $\mathcal{A}$ *is the action space,*
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ *is the transition probability function,*
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ *is the reward function,*
- $\gamma \in (0, 1]$ *is the discount factor.*

*The transition probability* $\mathcal{P}(s_k \mid s, a)$ *denotes the likelihood of reaching state* $s_k$ *after taking action* $a$ *in state* $s$*.*

Having defined the underlying Markov decision process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, it is now essential to describe how Deep-RL agents employ neural networks to parameterize both the policy and the value function. These components form the backbone of modern actor–critic methods.

The next subsection introduces the architectural design of these networks, outlining how the policy (actor) and value function (critic) are structured to initiate and enable continuous-control learning within a reinforcement learning framework.

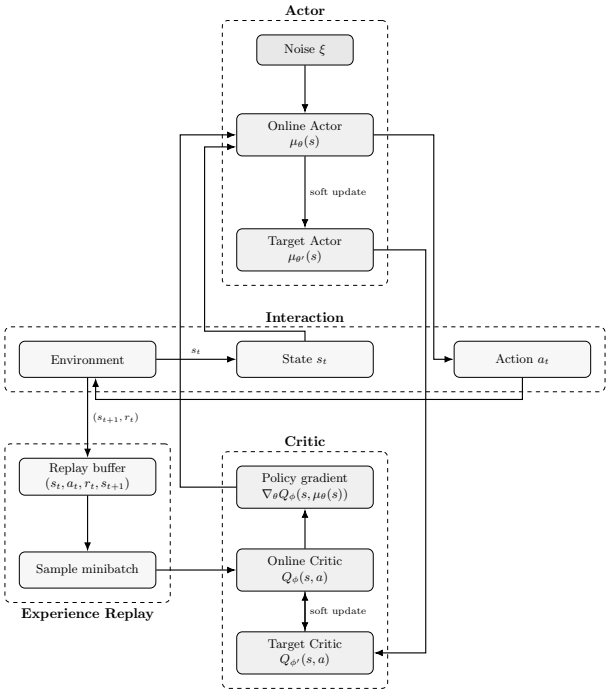



Figure 3.1: Original training architecture of DDPG in a vertical layout.

### 3.1.2 Policy Network Architecture

The actor network implements the deterministic policy as a deep neural network that outputs an action $a \in \mathcal{A}$, which in turn alters the agent's state $s$. Formally,

$$\pi_\theta(s) = \mathcal{F}_\theta(s), \tag{3.1}$$

where $\mathcal{F}_\theta$ is a feedforward neural network with parameters $\theta$, taking a state $s$ as input and producing a continuous action vector.

**Definition 3.2** (Policy)**.** *A policy $\pi_\theta$ is a mapping from states $\mathcal{S}$ to probability distributions over actions $\mathcal{A}$. Let $k$[1]$\in \mathbb{N}$denote a discrete decision time step along a trajectory. Given a state $s_k \in \mathcal{S}$ at step $k$, the policy specifies the probability of selecting action $a_k \in \mathcal{A}$:*

$$\pi_\theta(a_k \mid s_k) = \mathbb{P}[A_k = a_k \mid S_k = s_k]\,.$$

In DDPG, we specialize to deterministic policies $\pi_\theta^D : \mathcal{S} \to \mathcal{A}$, which map states directly to actions without sampling:

$$a = \pi_\theta^D(s).$$

The key distinction is that deterministic policies bypass action sampling entirely; the action is a direct functional output of the network rather than drawn from a distribution.

**State-Value Function** The value of a state under policy $\pi$ is defined as:

$$V^\pi(s) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \,\middle|\, S_0 = s\right]. \tag{3.2}$$

**Action-Value Function** The action-value function (Q-function) under policy $\pi$ is defined by:

$$Q^\pi(s,a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \,\middle|\, S_0 = s,\ A_0 = a\right]. \tag{3.3}$$

These functions are related by:

$$V^\pi(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}[Q^\pi(s,a)]\,. \tag{3.4}$$

[1]In most cases throughout this thesis, the index $k$ denotes the immediate successor transition of the current state–action pair, such that $k = t + 1$.

For deterministic policies, the expectation vanishes because the action is uniquely determined:

$$V^{\pi^D}(s) = Q^{\pi^D}(s,\, \pi^D(s)). \tag{3.5}$$

With these definitions established, we now formally introduce the critic network, which approximates the action-value function $Q^{\pi^D}(s,a)$ and forms the second component of the actor-critic architecture.

### 3.1.3 Critic Network and its Parameterization

**Definition 3.3** (Critic in DDPG)**.** *The* critic *is defined as the action-value function* $Q^{\pi^D}(s,a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$*, which estimates the expected cumulative discounted return obtained by taking action* $a$ *in state* $s$ *and subsequently following the deterministic policy* ${\pi^D}_\theta$*:*

$$Q^{\pi^D}(s,a) \;\approx\; \mathbb{E}_{s_k \sim \mathcal{P}(\cdot \mid s,a)} \left[ \sum_{k=0}^{\infty} \gamma^k \, r\Big(s_k,\, {\pi^D}_\theta(s_k)\Big) \right]. \tag{3.6}$$

*In DDPG, the critic learns this function using a neural network approximation.*

The Q-function is approximated by a parameterized neural network:

$$Q_\phi(s,a) \approx Q^{\pi^D}(s,a), \tag{3.7}$$

where $\phi$ denotes the critic network parameters. The critic takes both the state $s$ and action $a$ as inputs and outputs a scalar Q-value estimate.

#### Bellman Equations and Optimization

The Bellman equation characterizes the recursive relationship between the value of a state–action pair and the value of its successor states under a given policy. Understanding this structure is essential for later sections, where these relationships guide network optimization and enable more resource-efficient learning.

The Q-function defined in (3.3) satisfies:

$$Q^\pi(s,a) = \mathbb{E}_{s_k \sim \mathcal{P}(\cdot|s,a)} \Big[ \mathcal{R}(s,a) + \gamma \, \mathbb{E}_{a_k \sim \pi(\cdot|s_k)} [Q^\pi(s_k,a_k)] \Big]. \tag{3.8}$$

For deterministic policies, the Bellman equation simplifies as follows.

**Definition 3.4** (Bellman Equation for Deterministic Policies)**.** *Under a deterministic policy $\pi_\theta^D$ with $\theta$ parameters, the action-value function obeys:*

$$Q^{\pi^D}(s,a) = \mathbb{E}_{s_k \sim \mathcal{P}(\cdot|s,a)}\Big[r(s,a) + \gamma\, Q^{\pi^D}(s_k,\, \pi^D(s_k))\Big]\,. \tag{3.9}$$

*Here, $r(s,a)$ is the reward, $\gamma \in (0,1]$ is the discount factor, and $s_k \sim \mathcal{P}(\cdot|s,a)$ denotes sampling the next state from the transition distribution. This recursive equation defines $Q^{\pi^D}$ as the fixed point induced by repeatedly acting according to $\pi^D$.*

### 3.1.4 DDPG Actor–Critic Architecture

The **actor** $\pi_\theta^D(s)$ learns the deterministic policy, while the **critic network** $Q_\phi(s,a)$ learns the action-value function. Both are parameterized by neural networks with parameters $\theta$ and $\phi$, respectively.

#### Target Networks and Soft Updates

To improve stability, DDPG employs slowly updated target networks $Q_{\phi'}$ and $\pi_{\theta'}^D$. Their parameters are updated using soft (Polyak) averaging:

$$\begin{aligned} \phi' &\leftarrow \tau\phi + (1-\tau)\,\phi', \\ \theta' &\leftarrow \tau\theta + (1-\tau)\,\theta', \end{aligned} \tag{3.10}$$

where $0 < \tau \ll 1$ (typically 0.001).

The motivation stems from the fixed-point nature of the Bellman equation: Without target networks, the critic's regression target would change too rapidly, as it depends on parameters being updated in the same step, causing divergence or oscillation. Slowly updated targets provide a more stable learning signal, enabling convergence toward a moving but smooth objective.

#### Critic Learning

The critic learns by minimizing the TD error. The replay buffer $\mathcal{D}$ stores transitions $(s,a,r,s_k)$. A minibatch sampled from $\mathcal{D}$ is used to update the critic by minimizing:

$$L_{\text{critic}}(\phi) = \mathbb{E}_{(s,a,r,s_k)\sim\mathcal{D}}\Big[\Big(Q_\phi(s,a) - y\Big)^2\Big]\,, \tag{3.11}$$

where the target is computed via the target networks:

$$y = r + \gamma\, Q_{\phi'}\Big(s_k,\, \pi^D_{\theta'}(s_k)\Big)\,. \tag{3.12}$$

The gradient of the critic loss is:

$$\nabla_\phi L_{\text{critic}}(\phi) = \mathbb{E}_{(s,a,r,s_k)\sim\mathcal{D}}\Big[\nabla_\phi Q_\phi(s,a)\Big(Q_\phi(s,a) - y\Big)\Big]\,. \tag{3.13}$$

**Actor Learning**

The actor aims to select actions that maximize the critic's Q-value. Its objective is:

$$L_{\text{actor}}(\theta) = -\mathbb{E}_{s\sim\mathcal{D}}\Big[Q_\phi\Big(s,\, \pi^D_\theta(s)\Big)\Big] = J(\theta), \tag{3.14}$$

where the negative sign indicates we maximize expected return.

The deterministic policy gradient is:

$$\nabla_\theta L_{\text{actor}}(\theta) = -\mathbb{E}_{s\sim\mathcal{D}}\Big[\nabla_a Q_\phi(s,a)\Big|_{a=\pi^D_\theta(s)}\, \nabla_\theta \pi^D_\theta(s)\Big]\,, \tag{3.15}$$

which follows directly from the chain rule. This gradient instructs the actor how to adjust its parameters so as to choose actions that increase the critic's value estimate.

### 3.1.5 Computational Advantage

The deterministic policy gradient takes on a particularly simple mathematical form. It is expressed as the expected gradient of the action-value function:

$$\nabla_\theta J(\theta) = -\mathbb{E}_{s\sim\mathcal{D}}\Big[\nabla_a Q_\phi(s,a)\Big|_{a=\pi^D_\theta(s)}\, \nabla_\theta \pi^D_\theta(s)\Big]\,. \tag{3.16}$$

This formulation avoids integrating over the action space, making it significantly more computationally efficient for continuous-control problems. Such efficiency is valuable in our pursuit of developing reinforcement-learning agents that can be trained with reduced computational overhead.

With this foundation established, we now take a further step by applying DDPG in a constrained setting. This transition allows the mathematical structure of the method to unfold in a more advantageous manner, especially when examining optimization on tangential manifolds in constrained Markov decision processes

## 3.2 RL in ATACOM-based Constraint Manifold Approach

To apply a Deep-RL algorithm like DDPG in a CMDP setting, such as ATACOM, the agent is restricted to operate within equality and inequality constraints that define the feasible set of motions. These constraints limit the state-action trajectories to physically admissible regions, enabling safer exploration and inducing more structured gradient propagation.

**Definition 3.5** (Constrained Markov Decision Process)**.** *A Constrained Markov Decision Process (CMDP) is defined as an extension of a standard Markov Decision Process (MDP) by augmenting it with explicit constraint functions. Formally, a CMDP is represented by the tuple*

$$(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mathcal{C}),$$

*where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ denotes the action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition kernel, $\mathcal{R}$ is the reward function, and $\gamma \in (0, 1)$ is the discount factor. The set*

$$\mathcal{C} = \{c_i : \mathcal{S} \to \mathbb{R}\}_{i=1}^{m}$$

*collects $m$ immediate state-constraint functions that encode safety, physical, or task-specific limitations on admissible system behavior.*

In such settings, the deterministic policy ($\pi_\theta^D$) can no longer freely explore the entire action space. Instead, it must generate actions that satisfy feasibility conditions imposed by the system dynamics and task-specific constraints. These constraints are typically represented as smooth, differentiable functions that can be used in defining a constraint manifold $\mathcal{M}_c$. This notion of a manifold-based formulation is discussed further in the subsequent section.

**Definition 3.6** (Constraint Space Manifold)**.** *Let $\mathcal{Q} \subset \mathbb{R}^n$ denote the configuration space of a system, and let $c : \mathcal{Q} \to \mathbb{R}^m$ be a continuously differentiable mapping that collects $m$ independent equality constraint functions. The* constraint space manifold *is defined as the zero-level set*[2]*.*

$$\mathcal{M}_c = \{q \in \mathcal{Q} \mid c(q) = 0\}. \tag{3.17}$$

*If the Jacobian of the constraint function $J_c(q) = \partial c(q)/\partial q$ has full row rank for all $q \in \mathcal{M}_c$, then $\mathcal{M}_c$ is a smooth embedded sub-manifold of $\mathcal{Q}$ with dimension $n - m$.*

[2]The collection of all points where a given function evaluates to zero.

The manifold $\mathcal{M}_c$ defines a lower-dimensional feasible region within the ambient space $\mathcal{Q}$, with the dimensionality,

$$\dim(\mathcal{M}_c) = \dim(\mathcal{Q}) - \operatorname{rank}(J_c) = n - m.$$

The central challenge is to learn optimal policies while maintaining constraint satisfaction throughout the learning process. One way to enable policy learning is to restrict the state-action space to the tangential manifold of the constraint manifold; this is the approach used in ATACOM-based environments.

### 3.2.1 Review of ATACOM-based Constraint Manifold Approach

The ATACOM approach builds on the idea of CMDPs with constraint manifolds to learn a feasible policy in a manner that enables the learning process to operate within the tangential space of the constraint manifold. This effectively transforms the constrained learning problem into an unconstrained one, since all actions and state trajectories are confined to the manifold by construction.

To begin learning on the manifold using ATACOM, we first define the general constraint formulation it adopts. Consider a system with $Q$ configuration space and generalized coordinates $q \in \mathcal{Q} \subset \mathbb{R}^n$. The system is subject to a set of equality constraints

$$f : \mathcal{Q} \to \mathbb{R}^F,$$

and inequality constraints

$$g : \mathcal{Q} \to \mathbb{R}^G,$$

where both mappings are assumed to be twice continuously differentiable ($C^2$) and $F < Q$. The state constraints are defined as

$$f(q) = 0, \qquad g(q) \leq 0. \tag{3.18}$$

To convert the inequality constraints into equality constraints while preserving smoothness, a slack variable $\mu \in \mathbb{R}^G$ is introduced. This yields the augmented constraint mapping

$$c(q, \mu) = \begin{bmatrix} f(q) \\ g(q) + \frac{1}{2}\mu^2 \end{bmatrix} = 0, \tag{3.19}$$

where the square on $\mu$ is applied element-wise. The resulting constraint set defines a

smooth embedded manifold in the augmented space $\mathcal{Q} \times \mathbb{R}^G$ under the standard regularity assumptions.

#### 3.2.1.1 Tangent Space Construction and Action Parameterization

The tangent space of the constraint manifold characterizes all instantaneous motions that preserve constraint satisfaction. Taking the time derivative of the augmented constraint yields

$$\dot{c}(q, \mu, \dot{q}, \dot{\mu}) = J_c(q, \mu) \begin{bmatrix} \dot{q} \\ \dot{\mu} \end{bmatrix} = 0, \tag{3.20}$$

where the extended constraint Jacobian is given by

$$J_c(q, \mu) = \begin{bmatrix} J_f(q) & 0 \\ J_g(q) & \operatorname{diag}(\mu) \end{bmatrix} \in \mathbb{R}^{(F+G)\times(Q+G)}, \tag{3.21}$$

with $J_f(q) \in \mathbb{R}^{F\times Q}$ and $J_g(q) \in \mathbb{R}^{G\times Q}$ denoting the Jacobians of $f(q)$ and $g(q)$, respectively.

**Null Space Basis Construction** The null space of the extended constraint Jacobian defines the tangent space of the constraint manifold. Let

$$\mathcal{N}_c(q, \mu) = \operatorname{null}\Big(J_c(q, \mu)\Big) = \Big\{ v \in \mathbb{R}^{Q+G} \;\Big|\; J_c(q, \mu)\, v = 0 \Big\}. \tag{3.22}$$

An orthonormal basis for this null space is given by the matrix

$$N_c(q, \mu) \in \mathbb{R}^{(Q+G)\times(Q-F)},$$

which can be computed using singular value decomposition or QR factorization, such that

$$J_c(q, \mu)\, N_c(q, \mu) = 0. \tag{3.23}$$

Each column of $N_c(q, \mu)$ represents a basis vector of the tangent space of the constraint manifold. Feasible velocities on the manifold can therefore be parameterized as

$$\begin{bmatrix} \dot{q}_{\mathrm{T}} \\ \dot{\mu}_{\mathrm{T}} \end{bmatrix} = N_c(q, \mu)\, \alpha, \tag{3.24}$$

where $\alpha \in \mathbb{R}^{Q-F}$ denotes a set of free coordinates in the tangent space.

Substituting this parameterization into the constraint velocity equation yields

$$\dot{c}(q, \mu, \dot{q}_{\mathrm{T}}, \dot{\mu}_{\mathrm{T}}) = J_c(q, \mu)\, N_c(q, \mu)\, \alpha = 0, \tag{3.25}$$

which shows that the constraints remain satisfied regardless of the choice of $\alpha$. This property enables ATACOM to perform policy learning directly in the tangent space, ensuring feasibility by construction throughout the learning process.

#### 3.2.1.2 Acceleration-Level Action Construction

The ATACOM-based environment uses acceleration as the final input to the inverse-dynamic model to produce the action $a$ of the controlled entity. To reach the acceleration, we must go through the system constraint velocity as well.

In physical systems, it is often necessary to generate continuous velocity commands to ensure smooth, stable motion. Directly sampling velocities, however, does not guarantee continuity of the resulting trajectories. A common alternative is to sample accelerations, apply forces to the system, or recover velocities through numerical integration. When inequality constraints are present, an additional requirement arises whenever the system state lies on the boundary of a constraint, i.e., $g(q) = 0$, the corresponding constraint velocity should satisfy $\dot{g}(q, \dot{q}) \leq 0$ in order to prevent overshooting and constraint violation.

To address this, the original state constraints are converted into *viability constraints*, inspired by the linear viability condition. Specifically, the equality and inequality constraints are reformulated as

$$f(q) + K_f \dot{f}(q, \dot{q}) = f(q) + K_f J_f(q)\dot{q} = 0, \tag{3.26}$$

$$g(q) + K_g \dot{g}(q, \dot{q}) = g(q) + K_g J_g(q)\dot{q} \leq 0, \tag{3.27}$$

where $K_f \in \mathbb{R}^{F \times F}$ and $K_g \in \mathbb{R}^{G \times G}$ are diagonal matrices with strictly positive entries. These matrices regulate the allowable rates of change of the constraints relative to their current values. In particular, they bound the maximum admissible constraint velocities and determine how aggressively constraint violations are corrected when the system approaches or exceeds the constraint boundary.

Analogous to the slack-variable formulation used for the state constraints, the viability constraints are converted into an equality constraint set by introducing slack variables $\mu \in \mathbb{R}^G$:

$$c(q, \dot{q}, \mu) = \begin{bmatrix} f(q) + K_f J_f(q)\dot{q} \\ g(q) + K_g J_g(q)\dot{q} + \frac{1}{2}\mu^2 \end{bmatrix} = 0. \tag{3.28}$$

This formulation defines a constraint manifold in the augmented space $(q, \dot{q}, \mu)$, on which feasible system trajectories must evolve.

To maintain feasibility over time, the time derivative of the viability constraint is considered:

$$\dot{c}(q, \dot{q}, \ddot{q}, \mu, \dot{\mu}) = J_c(q, \mu) \begin{bmatrix} \ddot{q} \\ \dot{\mu} \end{bmatrix} + \psi(q, \dot{q}) = 0, \tag{3.29}$$

where the extended constraint Jacobian is given by

$$J_c(q, \mu) = \begin{bmatrix} K_f J_f(q) & 0 \\ K_g J_g(q) & \mathrm{diag}(\mu) \end{bmatrix}, \tag{3.30}$$

and the curvature-related term

$$\psi(q, \dot{q}) = \begin{bmatrix} J_f(q)\dot{q} + K_f b_f(q, \dot{q}) \\ J_g(q)\dot{q} + K_g b_g(q, \dot{q}) \end{bmatrix}. \tag{3.31}$$

The curvature terms are given by

$$b_f(q, \dot{q}) = \dot{q}^\top H_f(q)\dot{q}, \qquad b_g(q, \dot{q}) = \dot{q}^\top H_g(q)\dot{q},$$

where $H_f(q)$ and $H_g(q)$ denote the Hessians of $f(q)$ and $g(q)$, respectively. A feasible joint acceleration satisfying the constraint dynamics can then be constructed as

$$\begin{bmatrix} \ddot{q} \\ \dot{\mu} \end{bmatrix} = -J_c^\dagger(q, \mu)\, \psi(q, \dot{q}) + N_c(q, \mu)\, \alpha, \tag{3.32}$$

where $J_c^\dagger(q, \mu)$ denotes the Moore–Penrose pseudoinverse of the constraint Jacobian, $N_c(q, \mu)$ is a basis for its null space, and $\alpha$ represents a free acceleration coordinate.

### 3.2.2 Impact of Using ATACOM-based Approach with Sparse Networks.

The joint acceleration satisfying the viability and manifold-maintenance constraints is constructed and decomposed as

$$\begin{bmatrix} \ddot{q} \\ \dot{\mu} \end{bmatrix} = \underbrace{-J_c^\dagger(q, \mu)\, \psi(q, \dot{q})}_{\text{constraint maintenance}} + \underbrace{N_c(q, \mu)\, \alpha}_{\text{learned component}}. \tag{3.33}$$

where the first term enforces constraint maintenance by compensating for the curvature of the constraint manifold, while the second term lies entirely in the null space of $J_c(q, \mu)$ and represents the component learned by the policy. $J_c^\dagger(q, \mu)$ denotes the Moore–Penrose pseudoinverse of the extended constraint Jacobian and $N_c(q, \mu) \in \mathbb{R}^{(Q+G)\times(Q-F)}$ is a basis for the null space of $J_c(q, \mu)$ such that

$$J_c(q, \mu)\, N_c(q, \mu) = 0. \tag{3.34}$$

The vector $\alpha \in \mathbb{R}^{Q-F}$ parameterizes accelerations in the tangent space of the constraint manifold.

The first term,

$$-J_c^\dagger(q, \mu)\, \psi(q, \dot{q}), \tag{3.35}$$

is uniquely determined by the system state and enforces constraint consistency by compensating for the curvature of the constraint manifold. In contrast, the second term,

$$N_c(q, \mu)\, \alpha \;\in\; \mathcal{N}\Big(J_c(q, \mu)\Big), \tag{3.36}$$

lies entirely in the tangent space of the manifold and represents the only component of the acceleration that is influenced by the policy.

This decomposition is central to understanding why sparse networks excel in this framework. Rather than learning a mapping from states to accelerations in the full ambient space $\mathbb{R}^Q$, the policy learns a function

$$\pi_\theta^D : \mathcal{S} \to \mathbb{R}^{Q-F}, \qquad \pi_\theta^D(s) = \alpha, \tag{3.37}$$

where $Q - F$ is the intrinsic dimensionality of the constraint manifold. Since feasibility is guaranteed by construction through the pseudo-inverse term, the policy is not required to represent constraint avoidance or recovery behaviours. The learning problem is therefore restricted to optimizing motion along the manifold, i.e.,

$$\ddot{q}_{\text{learned}} = N_c(q, \mu)\, \pi_\theta^D(s). \tag{3.38}$$

As a consequence, the effective function class that must be represented by the policy is smooth, structured, and lower-dimensional than in unconstrained control settings. This reduction in representational complexity explains why heavily pruned networks are able to retain performance comparable to dense models: excess parameters that do not contribute to the tangent-space mapping $\alpha$ can be removed without degrading constraint satisfaction or task performance.

Essentially, Applying policy-based learning within this formulation forces the critic to learn value functions defined over the tangent space of the constraint manifold (denoted $\mathcal{T}_c$), rather than over the full configuration or action space, while the actor outputs control directions in a reduced-dimensional coordinate system $\alpha$, rather than directly in the full action space $\mathcal{A}$. This restructuring not only enforces constraint satisfaction by construction but also improves learning stability and accelerates convergence by yielding smoother and lower-variance gradients for both the actor and critic.

### 3.2.3 Actor–Critic Architecture in Constrained Settings

In a constraint-manifold setting, the actor–critic architecture must operate within the tangent-space action representation induced by ATACOM [30]. Rather than outputting a physical control input directly, the actor produces tangent-space coordinates

$$\alpha_k = \pi_\theta^D(s_k), \tag{3.39}$$

while the critic evaluates state–tangent pairs $(s, \alpha)$. The corresponding physical action (or acceleration-level command) applied to the environment is reconstructed through the manifold-consistent mapping

$$a_{\text{manifold}} = \Pi_{\mathcal{M}}(s, \alpha), \tag{3.40}$$

where $\Pi_{\mathcal{M}}$ denotes the ATACOM reconstruction (e.g., via null-space parameterization and constraint-maintenance terms), ensuring feasibility by construction.

#### Constrained-Action Bellman Relation

Defining the action-value function over tangent coordinates, $Q^{\pi^D}(s, \alpha)$, the Bellman equation becomes

$$Q^{\pi^D}(s, \alpha) = \mathbb{E}_{s_k \sim \mathcal{P}(\cdot|s, \Pi_{\mathcal{M}}(s,\alpha))}\Big[r\Big(s, \Pi_{\mathcal{M}}(s, \alpha)\Big) + \gamma\, Q^{\pi^D}\Big(s_k, \pi_\theta^D(s_k)\Big)\Big], \tag{3.41}$$

where $s_k \sim \mathcal{P}(\cdot \mid s, \Pi_{\mathcal{M}}(s, \alpha))$ is the next state. In practice, $Q_\phi(s, \alpha)$ is learned to approximate $Q^{\pi^D}(s, \alpha)$.

### Modified Loss Functions

The actor objective in tangent-space form is defined as

$$L_{\text{actor}}(\theta) = -\mathbb{E}_{s\sim\mathcal{D}}\Big[Q_\phi\Big(s, \pi_\theta^D(s)\Big)\Big]. \tag{3.42}$$

Taking the gradient yields the deterministic policy gradient in tangent space:

$$\nabla_\theta L_{\text{actor}}(\theta) = -\mathbb{E}_{s\sim\mathcal{D}}\left[\nabla_\theta \pi_\theta^D(s)\ \nabla_\alpha Q_\phi(s,\alpha)\Big|_{\alpha=\pi_\theta^D(s)}\right]. \tag{3.43}$$

The critic loss uses manifold-consistent targets:

$$L_{\text{critic}}(\phi) = \mathbb{E}_{(s,\alpha,r,s_k)\sim\mathcal{D}}\Big(Q_\phi(s,\alpha) - y_{\text{manifold}}\Big)^2, \tag{3.44}$$

with target

$$y_{\text{manifold}} = r + \gamma\, Q_{\phi'}\Big(s_k, \pi_{\theta'}^D(s_k)\Big), \tag{3.45}$$

where $\theta'$ and $\phi'$ denote slowly-updated target networks. Since the actor outputs tangent coordinates by design, $\pi_{\theta'}^D(s_k)$ remains in the tangent action space; any additional bounding or saturation of $\alpha$ can be applied directly in this coordinate space, as required by implementation.

## 3.3 Lottery Ticket Hypothesis in Constrained MDPs

The LTH, introduced by Frankle and Carbin [9], states that dense neural networks contain sparse sub-networks that can achieve comparable performance when trained in isolation. In the context of reinforcement learning, this hypothesis has important implications for both computational efficiency and the underlying dynamics of policy learning.

The existence of lottery tickets in Deep-RL can be intuitively explained through the notion of over-parameterization. Neural networks are universal function approximators and therefore often contain a significant amount of redundant capacity. For many MDPs, the optimal policy exhibits considerably lower intrinsic complexity than that suggested by standard deep network architectures. As a result, representational capacity is unevenly distributed across neurons in over-parameterized models, leading to substantial redundancy. When this excess capacity is removed through masking, the effective dimensionality of the optimization landscape is reduced. The modified gradient flows, formalized later in (3.55)

and (3.57), can therefore converge more rapidly and may settle into different, and in some cases better, local maxima than those reached by the corresponding dense networks.

**Definition 3.7** (Lottery Ticket Hypothesis for RL)**.** *A randomly initialized neural-network policy $\pi_\theta$ contains a sub-network $\pi_{\theta \odot m}$, where $m$ is a binary mask and $\odot$ denotes element-wise multiplication, such that when trained in isolation from the original initialization, it can match the performance of the full network [9].*

Consider a neural network $\mathcal{F}(x; \theta_0, m)$, where $\theta_0$ denotes the initial parameters and $m \in \{0,1\}^{|\theta|}$ is a binary pruning mask. A *winning ticket* is a tuple $(\theta_0, m)$ satisfying

$$\left| L_{\text{actor}}(\theta_0^* \odot \mathbf{1}) - L_{\text{actor}}(\theta_0^* \odot m) \right| \leq \varepsilon, \tag{3.46}$$

where $L_{\text{actor}}$ is the actor objective defined in (3.42), $\theta_0^* \odot \mathbf{1}$ and $\theta_0^* \odot m$ denote the trained dense and sparse parameters, respectively, and $\varepsilon$ is a small tolerance measuring proximity in performance.

### 3.3.1 Mathematical Framework for Lottery Tickets in RL

#### Network Sparsification

Given a neural network with parameters $\theta$, a binary mask $m \in \{0,1\}^n$ defines a sparse sub-network

$$\pi^D{}_{\text{sparse}}(s; \theta, m) = \pi^D(s; \theta \odot m), \tag{3.47}$$

The sparsity level associated with mask $m$ is given by

$$\text{Sparsity}(m) = 1 - \frac{\|m\|_0}{\theta}, \tag{3.48}$$

where $\|m\|_0$ is the number of nonzero entries in the mask $m$, and $\theta$ is the number of parameters in the dense network.

#### Unstructured Pruning

Unstructured pruning removes individual parameters from a neural network based on an importance criterion, such as weight magnitude, without enforcing any constraints on the

resulting sparsity pattern. Let $W \in \mathbb{R}^{m \times n}$ denote a weight matrix of a neural network layer. Unstructured pruning applies a binary mask $M \in \{0,1\}^{m \times n}$ to yield

$$W' = M \odot W, \tag{3.49}$$

where $\odot$ denotes the Hadamard (element-wise) product. The mask $M$ is typically constructed by retaining parameters whose importance scores exceed a threshold, e.g.,

$$M_{ij} = \begin{cases} 1, & |W_{ij}| \geq \tau, \\ 0, & \text{otherwise.} \end{cases} \tag{3.50}$$

This approach preserves the original network architecture and layer dimensions while reducing the number of nonzero parameters. However, because the sparsity is irregular, unstructured pruning does not generally translate into reduced inference-time computation on standard hardware.

### Structured Pruning

Structured pruning removes parameters according to predefined architectural units, such as neurons, channels, or filters, thereby modifying the network topology itself. Let $W \in \mathbb{R}^{m \times n}$ denote a weight matrix, and let $\mathcal{I} \subset \{1, \ldots, m\}$ index the set of retained output units. Structured pruning at the neuron level yields the reduced weight matrix

$$W' = W[\mathcal{I}, :], \tag{3.51}$$

where $W[\mathcal{I}, :]$ denotes the submatrix of $W$ obtained by retaining only the rows indexed by $\mathcal{I}$.

More generally, structured pruning removes groups of parameters indexed by a set $\mathcal{G}$ and can be expressed via a selection operator $\mathcal{P}_{\mathcal{G}}$:

$$W' = \mathcal{P}_{\mathcal{G}}(W), \tag{3.52}$$

where $\mathcal{P}_{\mathcal{G}}$ removes entire rows, columns, or blocks of $W$ according to the chosen structural grouping. Unlike unstructured pruning, this procedure produces smaller dense matrices and directly reduces inference-time computational cost.

### Iterative Magnitude Pruning

A widely used method for discovering winning tickets is the IMP [10]. The procedure repeatedly trains, prunes, and rewinds the network back to its initial parameters, as detailed in Algorithm 3.1.

**Algorithm 3.1** Iterative Magnitude Pruning for RL

**Input:** per-round pruning fraction $p \in (0, 1)$, pruning rounds $K$, training horizon $T$

1: Initialize dense parameters $\theta^0 \in \mathbb{R}^N$
2: Initialize mask $m^{(0)} \leftarrow \mathbf{1} \in \{0, 1\}^N$
3: **for** iteration $k = 0, 1, \ldots, K-1$ **do**
4: Train policy $\pi_{\theta^{(k)} \odot m^{(k)}}$ for $T$ episodes
5: Compute per-parameter magnitudes

$$\mathcal{M}^{(k)} = \{\, ||\theta_i^{(k)} \odot m_i^{(k)}|| : m_i^{(k)} = 1 \,\}, \text{where } i \in \{1, 2 \ldots N\}$$

6: Prune the lowest magnitudes $p\%$ of active parameters with the help of $\mathcal{M}^{(k)}$
7: Update mask:

$$m_i^{(k+1)} \leftarrow \begin{cases} 0, & \text{if } |\theta_i^{(k)}| \text{ is among the } \lfloor p_k \mathcal{M}_k \rfloor \text{ smallest and } m_i^{(k)} = 1, \\ m_i^{(k)}, & \text{otherwise.} \end{cases}$$

8: Reset surviving weights: $\theta^{(k+1)} = \theta_0 \odot m^{(k+1)}$
9: Apply the mask as $\theta = \theta^0 \odot m^{K-1}$.
10: **return** Winning ticket $(\theta_0, m^{(K-1)})$

The performance difference between the trained dense network and its sparse counterpart is

$$\Delta J(m) = J(\theta^*_{\text{full}}) - J(\theta^*_0 \odot m),$$

where $J(\cdot)$ denotes the expected return. A mask $m$ corresponds to a winning ticket if

$$\Delta J(m) \leq \varepsilon,$$

for a small tolerance $\varepsilon > 0$.

**One-Shot Magnitude Pruning**

As a computationally efficient alternative to iterative pruning, the One Shot Magnitude Pruning (OSMP) applies a single pruning operation after an initial training phase. Unlike IMP, which alternates between training and pruning over multiple rounds, OSMP identifies and removes low-importance parameters in a single step, followed by optional retraining of the resulting sparse network.

**Algorithm 3.2** One-Shot Magnitude Pruning for RL

**Input:** pruning fraction $p \in (0, 1)$, training horizon $T$

1: Initialize dense parameters $\theta^0 \in \mathbb{R}^N$
2: Initialize mask $m \leftarrow \mathbf{1} \in \{0, 1\}^N$
3: Train policy $\pi_{\theta^0}$ for $T$ episodes
4: Compute per-parameter magnitudes

$$\mathcal{M} = \{\, |\theta_i^0| : i \in \{1, 2, \ldots, N\} \,\}$$

5: Prune the lowest $p\%$ of parameters according to $\mathcal{M}$

6: Update mask:

$$m_i \leftarrow \begin{cases} 0, & \text{if } |\theta_i^0| \text{ is among the } \lfloor pN \rfloor \text{ smallest,} \\ 1, & \text{otherwise.} \end{cases}$$

7: Optionally retrain the sparse policy $\pi_{\theta^0 \odot m}$ for $T$ episodes
8: **return** Pruned network $(\theta^0, m)$

### 3.3.2 DDPG Network Architecture for Lottery Tickets with CSM

In standard DDPG, the actor and critic networks can be pruned independently:

$$\pi_\theta^D(s) \;\to\; \pi_\theta^D \odot m_A(s), \qquad Q_\phi(s, a) \;\to\; Q_{\phi \odot m_C}(s, a).$$

Specifically, the actor produces tangent-space coordinates

$$\alpha_k = \pi_\theta^D(s_k), \tag{3.53}$$

while the critic evaluates state–tangent pairs $(s, \alpha)$. When learning on the constraint

manifold via ATACOM, the joint optimization becomes

$$L_{\text{actor}}(\theta \odot m_A) = -\mathbb{E}_{s\sim\mathcal{D}}\Big[Q_{\phi\odot m_C}(s,\alpha)\Big], \tag{3.54}$$

where actions are parameterized in the tangent-space coordinates $\alpha = \pi_\theta^D \odot m_A(s)$. The corresponding modified policy gradient is defined as

$$\nabla_\theta J(\theta \odot m_A) = -\mathbb{E}_{s\sim\mathcal{D}}\left[\nabla_\theta \pi_\theta^D \odot m_A(s)\ \nabla_\alpha Q_{\phi\odot m_C}(s,\alpha)\Big|_{\alpha=\pi_\theta^D\odot m_A(s)}\right]. \tag{3.55}$$

For the critic, the loss must account for the geometry of the constraint manifold,

$$L_{\text{critic}}(\phi \odot m_C) = \mathbb{E}_{(s,\alpha,r,s_k)\sim\mathcal{D}}\left[\Big(Q_{\phi\odot m_C}(s,\alpha) - y_{\text{manifold}}\Big)^2\right], \tag{3.56}$$

with the gradient given by

$$\nabla_\phi L_{\text{critic}}(\phi \odot m_C) = \mathbb{E}_{(s,\alpha,r,s_k)\sim\mathcal{D}}\left[\nabla_\phi Q_{\phi\odot m_C}(s,\alpha)\ \Big(Q_{\phi\odot m_C}(s,\alpha) - y_{\text{manifold}}\Big)\right]. \tag{3.57}$$

Here,

$$y_{\text{manifold}} = r + \gamma\, Q_{\phi'}\Big(s_k,\ \pi_\theta^{D'}(s_k)\Big),$$

where $\pi_\theta^{D'}$ and $Q_{\phi'}$ denote the target actor and critic networks, respectively. Since the actor outputs tangent-space coordinates by construction, the target action $\pi_\theta^{D'}(s_k)$ remains within the feasible tangent-space domain.

## 3.4 Pruning Strategies

We now proceed to the pruning strategies and network-reconstruction procedures used to obtain the sparse ticket networks. These form the methodological foundation for constructing the compact policies evaluated in the subsequent experimental sections.

A good cusp of the work involves determining an effective pruning strategy to obtain the required mask. OSMP and IMP are both viable approaches used in modern experiments relating to LTH, but IMP has been shown to be a more efficient and robust method.

The rationale for using OSMP for the simpler Circular Track environment (Fig. 5.1(a)) is that this environment is characterized by lower dimensionality and a simpler optimal policy, making it well-suited to OSMP. Since pruning is performed in a single step, it avoids the repeated retraining cycles required by IMP. This makes OSMP a practical

choice wherever task simplicity permits, enabling instant development of a compact policy. Under these conditions, OSMP can aggressively eliminate less significant parameters while retaining performance comparable to the dense baseline. In contrast, more complex and higher-dimensional environments require the gradual refinement of IMP to preserve policy expressiveness and constraint robustness.

The traditional OSMP eliminates the smallest-magnitude parameters in a single step, which is often suboptimal for deep and complex networks but can achieve competitive accuracy for relatively low sparsity levels and simple architectures [4]. Similarly, IMP uses the OSMP principle iteratively over small pruning scales. It follows a schedule that repeatedly prunes the lowest-magnitude parameters, followed by fine-tuning (retraining) for a small number of epochs to recover performance. This cycle is repeated until the desired pruning level is reached.

These pruning paradigms are discussed in detail in Section 3.3.1. The reader is encouraged to review those sections for a more detailed treatment of these methods. Neural network pruning encompasses several methodologies in practice, with structured and unstructured pruning being the primary approaches. In this work, structured pruning is employed in combination with either IMP or OSMP, depending on the task complexity.

The *Trikaya* framework adopts *structured pruning* as the primary mechanism for shrinking a network. In this approach, the pruning mask removes entire neurons rather than individual weights or parameters. This produces agents that can truly be described as "shrunk" rather than "compressed" as shown in Fig. 3.2. A mask is constructed and applied to the network in such a way that the parameters corresponding to entire neurons are zeroed out and effectively removed. This justifies referring to the resulting model as a shrunk ticket network. Section 4.3 elaborates on ticket-model construction and provide additional insight into the methodology used to build ticket agents, including a formal theoretical foundation to clarify masking and pruning within the framework.

**Preliminary Considerations.** Consider a network with two hidden layers $\_h1, \_h2$. Let the input (state) be $s \in \mathbb{R}^{d_s}$, hidden sizes/features $[n_1, n_2]$, and output (action) $a \in \mathbb{R}^{d_a}$, where $d_s$ and $d_a$ represent the dimensions of the state and action spaces, respectively. The weights and biases are

$$W_1 \in \mathbb{R}^{n_1 \times d_s}, \quad b_1 \in \mathbb{R}^{n_1}, \quad W_2 \in \mathbb{R}^{n_2 \times n_1}, \quad b_2 \in \mathbb{R}^{n_2}, \quad W_3 \in \mathbb{R}^{d_a \times n_2}, \quad b_3 \in \mathbb{R}^{d_a}.$$

**Making Masks for Layers.** For each masked layer $\ell \in \{1, 2\}$, we define a binary *row mask* $m_\ell \in \{0, 1\}^{n_\ell}$ and its diagonal matrix $M_\ell = \mathrm{diag}(m_\ell)$, with $M_\ell \in \mathbb{R}^{n_\ell \times n_\ell}$. Applying

a row mask *materializes* as

$$\tilde{W}_\ell \;=\; M_\ell\, W_\ell, \qquad \tilde{b}_\ell \;=\; M_\ell\, b_\ell, \tag{3.58}$$

where

$$\tilde{W}_\ell \in \mathbb{R}^{n_\ell \times d_s}, \quad \tilde{b}_\ell \in \mathbb{R}^{n_\ell}.$$

The diagonal matrix $M_\ell$ condenses row-wise multiplication into a single operation. With this, entire rows (neurons) and their corresponding biases are consistently zeroed.

**Row-Score and Mask Construction (LN-structured).** Let $W_\ell[i,:]$ denote row $i$ of $W_\ell$, i.e., the weights of the $i$-th neuron in layer $\ell$. Define the row score $s_\ell(i)$ as

$$s_\ell(i) \;=\; \|W_\ell[i,:]\|_2\,. \tag{3.59}$$

**Iterative Prune Schedule to Acquire Target Sparsity.** With $R$ rounds aimed at achieving a target sparsity $\zeta$, we choose the per-round prune fraction $q$ such that

$$q \;=\; 1-(1-\zeta)^{1/R}, \tag{3.60}$$

where $(1-q)$ is the fraction left alive in each round and $(1-\zeta)$ is the final surviving fraction after all pruning. After $R$ rounds, the alive fraction satisfies

$$\overbrace{(1-q)\cdot(1-q)\ldots(1-q)}^{R \text{ times}}\cdot(1) = (1-q)^R = (1-\zeta),$$

which yields

$$q \;=\; 1-(1-\zeta)^{1/R}.$$

After each round, pruning a fraction $q$ of neurons from each layer, the network is trained for $K$ epochs before applying the next mask update.

**Training-time Masked Constraint Optimization.** To maintain zeros during learning, one can enforce either of the following equivalent constraints:

$$\begin{aligned} &\text{(re-projection)} && W_\ell \leftarrow M_\ell W_\ell,\;\; b_\ell \leftarrow M_\ell b_\ell \quad \text{after each optimizer step;} \\ &\text{(gradient masking)} && \nabla_{W_\ell} \leftarrow M_\ell \odot \nabla_{W_\ell},\;\; \nabla_{b_\ell} \leftarrow m_\ell \odot \nabla_{b_\ell}, \end{aligned}$$

where $\odot$ denotes the Hadamard product. Using either approach ensures that pruned parameters remain identically zero throughout training.

**(For one-shot) LN-structured Row Pruning with Bias Zeroing.** For environments handled with one-shot pruning[3], each layer $\ell \in \{1, 2\}$ computes row scores using the $\ell_2$ norm $\left(\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}\right)$:

$$s_\ell(i) \;=\; \|W_\ell[i,:]\|_2\,, \qquad i = 1, \ldots, n_\ell.$$

Given a sparsity fraction $q \in (0, 1)$, we prune a fraction $q$ of rows with the smallest scores:

$$\mathcal{P}_\ell \;=\; \underset{\substack{\mathcal{I}\subset\{1,\ldots,n_\ell\}\\ |\mathcal{I}|=\lfloor qn_\ell\rfloor}}{\operatorname{argmin}} \sum_{i\in\mathcal{I}} s_\ell(i).$$

Define the binary *row mask* $m_\ell \in \{0, 1\}^{n_\ell}$ by

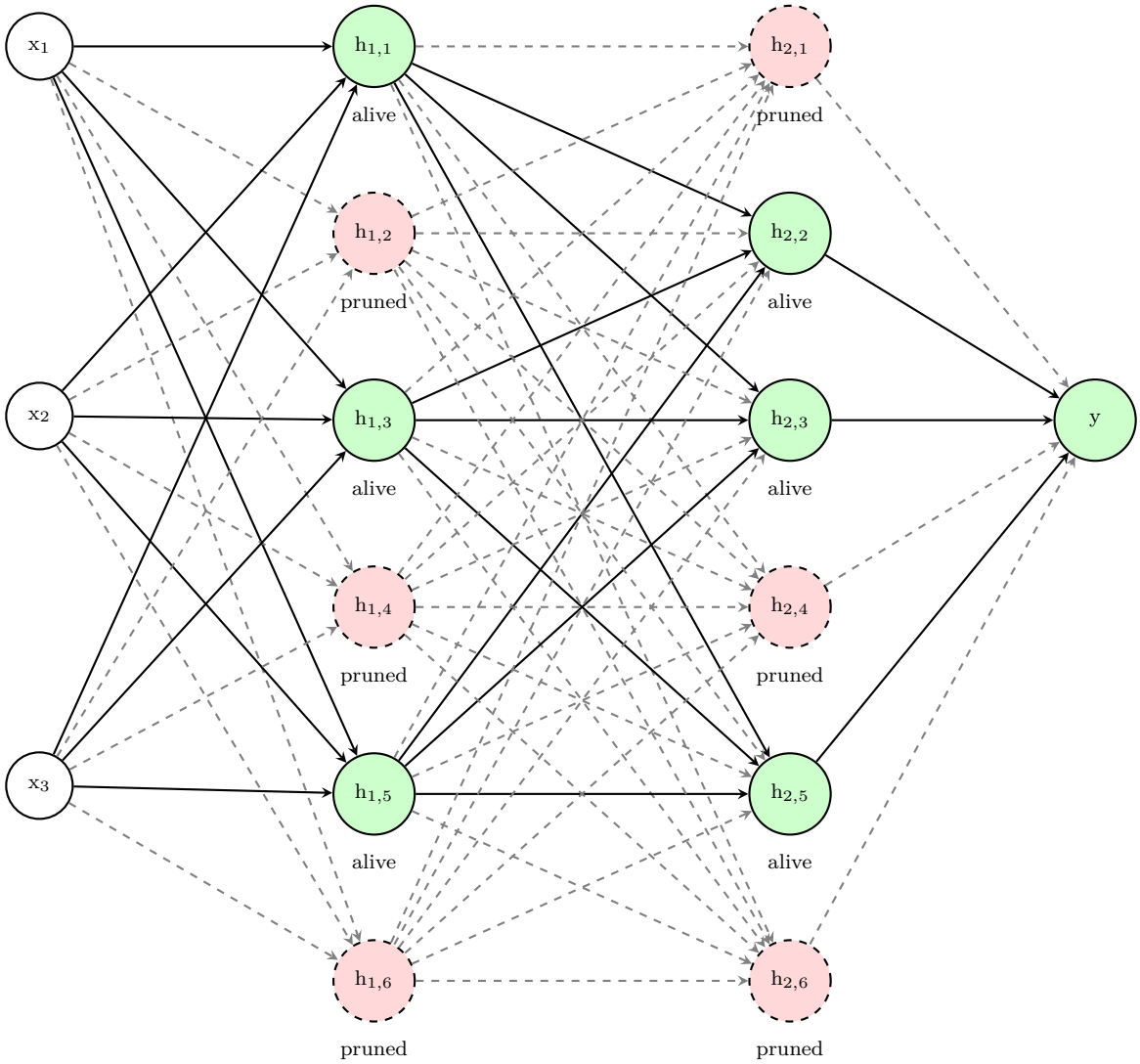
$$m_\ell(i) = \begin{cases} 0, & i \in \mathcal{P}_\ell, \\ 1, & \text{otherwise}, \end{cases} \qquad M_\ell \;=\; \operatorname{diag}(m_\ell).$$

Applying the mask yields the masked parameter.s

$$\tilde{W}_\ell \;=\; M_\ell W_\ell, \qquad \tilde{b}_\ell \;=\; M_\ell b_\ell.$$

If the biases are included in scoring, they are zeroed by the same row mask. $b_\ell \leftarrow m_\ell \odot b_\ell$.

[3]The one-shot pruning paradigm is exclusively used to tackle only the simplest Circular Track environment (refer to Fig. 5.1(a)). The main reason for using one-shot instead of IMP is to reduce computational overhead when employing the *Trikaya* framework to produce tickets.




$\mathcal{S}_1 = \{1, 3, 5\}$ (survivors in first hidden layer)

$\mathcal{S}_2 = \{2, 3, 5\}$ (survivors in second hidden layer)

Figure 3.2: Two hidden layers with six neurons each. In _h1, neurons 1, 3, and 5 survive ($\mathcal{S}_1 = \{1, 3, 5\}$). In _h2, neurons 2, 3, and 5 survive ($\mathcal{S}_2 = \{2, 3, 5\}$). Dashed red nodes and connections are pruned. Survivors define the ticket network.

## 3.5 Summary

This chapter presented the key ideas underlying the *Trikaya* framework in a structured and coherent manner. It began by establishing the mathematical foundations of reinforcement learning, including Markov decision processes, policy and value function formulations, and the actor–critic architecture underlying DDPG. These preliminaries provided the baseline against which subsequent constrained formulations were developed. The discussion was then extended to constrained MDPs through the ATACOM-based constraint manifold approach, introducing tangent-space action parameterization, null-space construction, and acceleration-level control, and highlighting the implications of operating on a constraint space manifold.

Building on this formulation, the chapter examined actor–critic learning in constrained settings, including the constrained-action Bellman relation and the corresponding modifications to actor and critic loss functions. The theoretical groundwork was then used to motivate the application of the LTH to constrained decision-making problems. In this context, the chapter explained how sparse subnetworks, or tickets, can benefit from learning within an action space embedded in a tangential manifold, thereby exploiting the geometric structure imposed by the CSM.

Finally, the chapter detailed the pruning strategies and reconstruction procedures used to obtain these ticket networks, including iterative magnitude pruning, structured mask construction, and training-time masked optimization, culminating in a unified methodology for deploying lottery ticket networks within constrained reinforcement learning settings.

# Chapter 4

# Trikaya Framework

We now present the main contributions of the research in this thesis and provide compiled algorithms for implementing the *Trikaya* framework, thereby enabling the training of shorter models with comparable performance metrics.

In this chapter, Section 4.1 states the central proposed hypothesis that motivates the framework. Section 4.2 then discusses the expected benefits of applying the framework, presenting mathematical and logical expressions to convey the advantages of reduced computational overhead. Section 4.3 further details the proposed procedure for building and training ticket networks, which are shortened models that can compete with fully dense networks while using only a fraction of the memory and computational overhead. Finally, in Section 4.4 a summary is provided with the main points presented in the chapter.

## 4.1 Central Hypothesis

The central hypothesis of this thesis emerged from repeated empirical observations drawn from training metrics across the control tasks studied in constraint-aware ATACOM environments. One plausible explanation is that once the environment restricts feasible motion to a low-dimensional null/tangential subspace, the policy no longer needs to represent arbitrary actions in the complete state space. This geometric restriction implicitly filters out many "violation directions" and concentrates learning on a structured set of admissible degrees of freedom.

Since sparsification removes redundant capacity and limits how a network can distribute representational power, we consider the combination of Constraint Space Manifold

alignment and Lottery Ticket Hypothesis in a harmonious mechanism. Manifold geometry narrows the effective action space, while sparsity encourages the policy to allocate its remaining capacity along those exact feasible directions. This impact is also mathematically discussed in the Section 3.2.2. Finally, we put forward the following hypothesis.

**Hypothesis.** *The Lottery Ticket Hypothesis becomes more effective in identifying winning-ticket networks for reinforcement learning problems involving high-dimensional control tasks when combined with environments that enforce an action space aligned with the Null/ Tangential space of the Constraint Space Manifold (such as ATACOM environments).*

This hypothesis motivates the two primary contributions of this chapter: Section 4.2, which explains why constraint-manifold structure strengthens ticket discovery and efficiency, and Section 4.3 details the practical procedure for constructing and training the resulting ticket networks.

## 4.2 Benefits of Lottery Tickets in Constraint Manifold Settings

A significant connection emerges between lottery-ticket sparsity and the geometry of constraint manifolds. In particular, the low-dimensional structure of tangential action spaces naturally aligns with the sparse connectivity patterns discovered through pruning. This section formalizes why sparse sub-networks (winning tickets) tend to perform exceptionally well when learning on the constraint manifold. In this section, we present mathematical benefits expressed through equations derived using the Trikaya framework.

### 4.2.1 Proposed Mathematical Computational Benefits of Learning Lottery Tickets on CSM

#### 4.2.1.1 Tangent Space Basis Alignment

In unconstrained settings, neural networks must represent policies over the full action space $\mathcal{A}$, requiring dense connectivity to model high-dimensional decision boundaries. In contrast, constrained environments restrict feasible motion to the tangent space $\mathcal{N}_c$, whose dimension is

$$d_T = \dim(\mathcal{Q}) - \operatorname{rank}(J_c(q))$$

Often, it is much smaller than the ambient space.

This dimensional reduction induces a natural compatibility with sparse architectures:

$$\text{Network Capacity Required} \propto \dim(\mathcal{N}_c) \ll \dim(\mathcal{A}). \tag{4.1}$$

Sparse lottery-ticket networks therefore learn representations that inherently align with the reduced-degree-of-freedom structure of the tangent space.

**Dimensional Compatibility** The null-space matrix $N_c(q)$ provides a canonical decomposition of admissible action directions. Winning-ticket masks discovered under constraint-aware training exhibit structured connectivity patterns that align with the manifold geometry:

$$\text{Winning Mask Pattern} \subset \text{Span}(N_c^\top). \tag{4.2}$$

This helps enabling the following characteristics within our results:

- *Structured pruning patterns*: pruned parameters correspond to directions orthogonal to the manifold.
- *Preserved critical paths*: surviving connections encode tangent-space dynamics.

These structured patterns help explain why sparse networks remain performant even at extreme sparsity levels when operating in constrained environments.

### Reduced Forward-Pass Complexity

Sparse networks trained on constraint manifolds exhibit multiplicative efficiency gains arising from both parameter reduction and reduced effective action dimensionality. For sparse networks,

$$\text{Complexity}_{\text{sparse, constrained}} = O(\zeta \cdot |\theta| \cdot d_T),$$

where $\zeta$ is the sparsity level, $|\theta|$ is the parameter count, and $d_T$ is the tangent-space dimension. For dense unconstrained networks,

$$\text{Complexity}_{\text{dense, unconstrained}} = O(|\theta| \cdot \dim(\mathcal{A})).$$

Thus, the approximate speedup factor is

$$\text{Speedup Factor} = \frac{1}{\zeta} \cdot \frac{\dim(\mathcal{A})}{d_T}. \tag{4.3}$$

Even in moderately constrained environments, this factor can exceed an order of magnitude, which explains the improved training-time performance observed (later) in experiments.

### Memory Efficiency and Parameter Reduction

Sparse actor–critic architectures on constraint manifolds reduce several memory burdens simultaneously.

**Parameter Memory** For sparsity levels $\zeta_A$ (actor) and $\zeta_C$ (critic),

$$M_{\text{parameters}} = \zeta_A|\theta_A| + \zeta_C|\theta_C|. \tag{4.4}$$

**Forward-Pass Memory** Sparse connectivity reduces the storage required for activations and intermediate tensors. The forward-pass memory corresponding to the actor and critic networks is defined as follows:

$$M_{\text{actor,forward}} = \sum_{l=1}^{L_A} \left(\zeta_A^{(l)} n_l n_{l-1} + \zeta_A^{(l)} n_l\right), \tag{4.5a}$$

$$M_{\text{critic,forward}} = \sum_{l=1}^{L_C} \left(\zeta_C^{(l)} n_l n_{l-1} + \zeta_C^{(l)} n_l\right), \tag{4.5b}$$

where $n_l$ denotes the number of neurons in layer $l$, and $s^{(l)}$ is the layerwise sparsity.

**Gradient Memory** Gradients are stored only for the remaining active parameters. The memory they occupy is given by

$$M_{\text{gradients}} = \zeta_A|\theta_A| + \zeta_C|\theta_C|. \tag{4.6}$$

The total memory-efficiency factor is then

$$\eta_{\text{memory}} = \frac{M_{\text{dense}}}{M_{\text{sparse}}} = \frac{1}{\bar{\zeta}}, \tag{4.7}$$

where $\bar{\zeta}$ is the average sparsity across both networks.

This efficiency enables longer rollouts, larger batch sizes, and reduced GPU/CPU memory usage, which is particularly valuable in constrained RL, where additional Jacobian and projection computations are often required.

### 4.2.2 Key Logical Computational Benefits of Learning Lottery Tickets on CSM

#### 4.2.2.1 Evaluation Speedup

ATACOM enforces feasibility by projecting candidate actions onto the tangent space using the constraint Jacobian $J_c(q, \mu)$ and its null-space basis $N_c(q, \mu)$. While this geometric projection is independent of the neural network architecture, sparsity in the actor and critic networks reduces the computational cost of evaluating the policy $\pi_\theta^D(s)$ and the value function $Q_\phi(s, \alpha)$. In practice, this leads to faster forward and backward passes during training, while preserving the exact same constraint projection and feasibility guarantees.

Pruning reduces the computational load of the actor and critic networks, which lowers the overall cost of each training step. Although the geometric projection defined by $J_c(q, \mu)$ and $N_c(q, \mu)$ is unchanged, the reduced neural-network workload decreases the total wall-clock time per iteration, leading to faster training in practice while preserving feasibility guarantees.

#### 4.2.2.2 Gradient Alignment and Numerical Stability

Since the physical action is reconstructed as

$$a_{\text{manifold}} = N_c(q, \mu)\, \alpha + (\text{constraint-maintenance term}),$$

the gradient passed from the critic to the actor satisfies

$$\nabla_\alpha Q_\phi(s, \alpha) = N_c(q, \mu)^\top \nabla_a Q_\phi(s, a).$$

This implies that the policy gradient is projected onto the tangent space of the constraint manifold, and therefore contains no components in constraint-violating directions. As a result, the actor is updated only along feasible degrees of freedom, leading to more stable and well-conditioned learning dynamics near constraint boundaries.

#### 4.2.2.3 Robustness to Approximation Errors

Sparse networks within tangential action spaces exhibit *inherent robustness* to constraint violations. Because pruning reduces representational degrees of freedom, the model is less able to produce actions that deviate from feasible directions. This induces smaller constraint violations, smoother value-function gradients, and more stable behaviour near constraint boundaries.

**Sparse Network Constraint Robustness** Since actions are generated in the tangent space, pruning effectively removes directions that are orthogonal to the manifold. That is, $\max_i |c_i(q, \mu_{\text{ticket}}(q))|$ is reduced by sparsity. With fewer parameters available to push the policy into infeasible regions, sparse networks naturally maintain lower constraint violations.

**Implicit Regularization Effects** The ATACOMs tangent-space formulation embeds constraint consistent behaviour directly into the environment. This approach interacts harmoniously with sparsity, where pruning removes redundant paths, while the manifold geometry ensures that the surviving paths remain aligned with feasible regions. Thus, ATACOM *stabilizes* lottery-ticket subnetworks and promotes the discovery of consistent, constraint-respecting sparse policies.

#### 4.2.2.4 Memory Efficiency in Constrained Learning

Constrained RL typically requires multiple networks (actor, critic, constraint evaluators). Sparse subnetworks significantly reduce memory across all components, improving cache locality and enabling the same training budget to support:

- Larger batch sizes
- Longer rollout horizons
- Deeper or wider architectures at equal memory cost.

This efficiency opens the door to deploying large, high-capacity models on low-resource hardware, which is an essential step toward practical real-world deployment.

## 4.3 Building and Training Trikaya Ticket Networks

This section provides an in-depth overview of how to build the ticket networks. It follows the theoretical foundation laid in Section 3.4 to construct a clear understanding of the ticket-building and pruning process.

Initially, we train the original dense network using Algorithm 4.1. Let the trained dense actor network have two masked hidden layers $\_h1, \_h2$ and an output layer $\_h3$, which have the following parameters (Weights and biases trained using Algorithm 4.1):

$$W_1 \in \mathbb{R}^{n_1 \times d_s}, \quad b_1 \in \mathbb{R}^{n_1}, \qquad W_2 \in \mathbb{R}^{n_2 \times n_1}, \quad b_2 \in \mathbb{R}^{n_2}, \qquad W_3 \in \mathbb{R}^{d_a \times n_2}, \quad b_3 \in \mathbb{R}^{d_a}.$$

After all parameters have been trained, we use the trained network and its parameters to construct the ticket network that follows the *Trikaya* framework, as described in Algorithm 4.4.

Additionally, as discussed in Section 2.3.3, at the reset step, a late-rewinding approach [10] is used to re-initialize the ticket network. The late rewind is performed to restore the network state to *epoch*=1 in the original dense training, stated in Algorithm 4.1. After loading the untrained *epoch* 1 dense agent reference, the *initial state* is cached as a checkpoint for rewinding. Let the loaded untrained dense agent (at *epoch*=1) have parameters

$$W_\ell^{(t)},\ b_\ell^{(t)} \quad (\ell = 1, 2, 3), \quad t = 1, \tag{4.8a}$$

$$\theta^{(0)} = \Big\{ \underbrace{(W_1^{(1)}, b_1^{(1)})}_{\text{Layer 1}},\ \underbrace{(W_2^{(1)}, b_2^{(1)})}_{\text{Layer 2}},\ \underbrace{(W_3^{(1)}, b_3^{(1)})}_{\text{Layer 3}} \Big\}, \tag{4.8b}$$

$$\theta^{(0)} = \left\{ W_\ell^{(1)},\ b_\ell^{(1)} \right\}_{\ell=1}^{3}. \tag{4.8c}$$

Here, $\theta^{(0)}$ is the vector that contains the initial parameters used to perform re-initialization when creating the ticket agent. A mask $m_\ell$ is applied to $\theta^{(0)}$ to only keep the surviving parameters which are $\theta' = m_\ell \odot \theta^{(0)}$, where $\theta' \subseteq \theta^{(0)}$. We now proceed to how we construct the mask $m_\ell$.

To build the ticket network, we need to construct a mask $m_\ell$ using Iterative Magnitude Pruning in a structured pruning manner; both approaches are discussed in detail in

**Algorithm 4.1** Baseline–Full Training (DDPG)

**Input:** Environment Parameters ($\mathcal{E}$)
▷ *env. specific parameters (horizon, gamma, timesteps, etc.)*
Algorithm Parameters ($\kappa$)
Training Parameters ($\mathcal{T}$)
$[n_1, n_2]$ Hidden Layer Features (`n_features`).

**Output:**
Trained full agent, best metrics.

▷ *Initializations:* ◁
$\mathcal{M} \leftarrow$ MDP $\leftarrow (\mathcal{E})$
$\Omega(\boldsymbol{\pi}, \boldsymbol{Q}) \leftarrow$ Agent $\leftarrow (\mathcal{M}, \texttt{n\_features}, \kappa)$
$\mathcal{M}(S) \leftarrow \mathcal{P}_{minmax}(\mathcal{M}(S))$
▷ *minmax-preprocessor $\mathcal{P}_{minmax}$ normalizes observation states $S$ of the environment*
Compute the metrics for initialized agent (**J, R**)
$J \leftarrow \texttt{mean}(\textsc{ComputeMetrics}(\Omega_{\mathcal{D}=\{s_i,a_i,r_i,s'_i\}_{i=1}^{N}}, \mathcal{E}_\gamma))$
$R \leftarrow \texttt{mean}(\textsc{ComputeMetrics}(\Omega_{\mathcal{D}=\{s_i,a_i,r_i,s'_i\}_{i=1}^{N}}, 1.0))$
▷ *mean discounted reward & reward ref. Algorithm 4.2*
Update best$_{\{\mathbf{J},\ \mathbf{R}\}} \leftarrow$(**J, R**)

**save** checkpoint agent $\Omega^0_{Full}$ (*at epoch=0*)
▷ *Training loop* ◁
**for** *each epoch i* **do**
  Train $\Omega^{i+1}_{Full} \leftarrow \textsc{Learn}(\mathcal{M}, \mathcal{E}, \mathcal{T}, \Omega^i_{Full})$ ▷ *Learn uses Algorithm Algorithm 4.3*
  Compute the metrics for agent (**J, R**)
  **if J** > best$_{\{\mathbf{J}\}}$ **then**:
    Update best$_{\{\mathbf{J},\ \mathbf{R}\}} \leftarrow$(**J, R**)
    **save** best agent($\Omega^i_{Full}$) (*at epoch=i*)
  **if** $i \leq 2$ **then**:
    **save** checkpoint agent($\Omega^{i\leq 2}_{Full}$) (*at epoch=i*)
    ▷ *For creating a late-rewind agent checkpoint*
  **return** metrics **J, R**

Section 3.3.1. Pruning follows the schedule defined in (3.60).

We initialize the mask $m_\ell$ as a vector of ones. Let $m_\ell^{(r)}$ denote the current layer mask at round $r$, and let $\mathcal{A}_\ell^{(r)} = \{i \mid m_\ell^{(r)}(i) = 1\}$ be the set of alive rows (neurons) at round

**Algorithm 4.2** ComputeMetrics

**Input:** Dataset $\mathcal{D} = \{s_i, a_i, r_i, s'_i\}_{i=1}^{N}$, discount factor $\gamma \in [0, 1]$
**Output:** Mean discounted return $J$, mean return $R$

```
function ComputeMetrics(D, γ)
    Initialize J_episode ← 0, steps ← 0, J ← [ ]
    for i = 1 to |D| do  ▷ |D| represents cardinality used for getting number of entries
        J_episode ← J_episode + γ^steps · r_i
        steps ← steps + 1
        if i = |D| then
            Append J_episode to J
            J_episode ← 0; steps ← 0
    if |J| = 0 then                                                    ▷ Same as above
        return [0]
    else
        return J
```

$r$. We compute the raw scores on the *underlying weight tensor* required for updating the mask:

$$s_\ell^{(r)}(i) = \begin{cases} \|W_\ell^{(r-1)}[i,:]\|_2, & \text{(default, } p\text{=2)}, \\ \|W_\ell^{(r-1)}[i,:]\|_p, & \text{(for a general } p \geq 1\text{; e.g. } p\text{=1)}. \end{cases}$$

If the bias is present and included in scoring, the raw scores are adjusted to include a bias term that updates the score to $s_\ell^{(r)}(i) \leftarrow \sqrt{\left(s_\ell^{(r)}(i)\right)^2 + |b_\ell^{(r-1)}(i)|^2}$. The $r-1$ shows that the current round's score is computed from the previous round's weights and biases. Given the per-round prune fraction $q \in (0, 1)$, we define the integral number of neurons to be pruned as $k_\ell^{(r)}$, where

$$k_\ell^{(r)} = \begin{cases} \left\lceil q \cdot \max(|\mathcal{A}_\ell^{(r)}|, 1) \right\rceil, & \text{if } |\mathcal{A}_\ell^{(r)}| > 1, \\ 0, & \text{if } |\mathcal{A}_\ell^{(r)}| = 1. \end{cases}$$

Then the neurons to be pruned with the smallest $s_\ell^{(r)}(i)$ score are pruned to obtain the updated mask $m_\ell^{(r+1)}$. Rows already pruned are excluded by setting $s_\ell^{(r)}(i) = +\infty$ for $i \notin \mathcal{A}_\ell^{(r)}$ to avoid repeated pruning of the same rows.

After applying the mask $m_\ell^{(r+1)}$, the new masked weights satisfy

$$\tilde{W}_\ell^{(r+1)} = M_\ell^{(r+1)} W_\ell^{(r)},$$

**Algorithm 4.3** LEARN function based on ATACOM [30]

**Input:** MDP $\mathcal{M}$,
Agent $\Omega(\boldsymbol{\pi}, \boldsymbol{Q})$,
Environment Parameters ($\mathcal{E}$),
Training Parameters ($\mathcal{T}$).
**Output:** Trained $\Omega(\boldsymbol{\pi}, \boldsymbol{Q})$ by an epoch.

**function** LEARN($\mathcal{M}, \mathcal{E}, \mathcal{T}, \Omega(\boldsymbol{\pi}, \boldsymbol{Q})$)
  ▷ *Extract the constraint, scaling and training parameter sets* ◁
  Constraints set $\{f,\ g,\ J_f,\ J_g,\ b_f,\ b_g\} \leftarrow \mathcal{M}_{\{f, g, J_f, J_g, b_f, b_g\}}$
  Scaling set $\{K_c,\ K_f,\ K_g\} \leftarrow \mathcal{E}_{\{K_c,\ K_f,\ K_g\}}$
  $\Delta T \leftarrow \mathcal{T}_{\{\Delta T\}}$
  **for** each episode **do** ▷ *follows ATACOM Algorithm 2.1*
    Reset to initial feasible state $s_0$, slack variable $\mu_0$. ▷ $\mu_0$ *is for the inequality constraint* $g$
    **for** each time step $k$ **do**
      Sample policy actions $\alpha_k \sim \pi(\cdot \mid s_k)$ from $\Omega(\boldsymbol{\pi}, \boldsymbol{Q})$. ▷ *Coordinate* $\alpha_k$ *acts as nearby actions at state* $s_k$
      Observe $q_k,\ \dot{q}_k$ from $s_k$.
      Compute $J_{c,k} = J_c(q_k, \mu_k), \quad \psi_k = \psi(q_k, \dot{q}_k), \quad c_k = c(q_k, \dot{q}_k, \mu_k)$.
      Compute the RCEF null-space/tangent basis $N_c^R$.
      Compute tangent-space acceleration
      $\begin{bmatrix} \ddot{q}_k \\ \dot{\mu}_k \end{bmatrix} \leftarrow -J_{c,k}^{\dagger}\Big[K_c\, c_k + \psi_k\Big] + N_c^R\, \alpha_k$. ▷ *Performs computation using constraint and scaling sets*
      Clip joint acceleration $\ddot{q}_k \leftarrow \text{clip}(\ddot{q}_k, a_l, a_u)$.
      Integrate slack: $\mu_{k+1} = \mu_k + \dot{\mu}_k\, \Delta T$.
      Apply control $a_k = \Lambda(\ddot{q}_k)$ to the environment.
      Observe next state $s_{k+1}$ and reward $r_k$.
      Use the transition tuple $(s_k, a_k, s_{k+1}, r_k)$ for fit training actor-critic networks of the agent $\Omega(\boldsymbol{\pi}, \boldsymbol{Q})$.
  **return** $\Omega(\boldsymbol{\pi}, \boldsymbol{Q})$ trained by an epoch.

Where $M_\ell$ is a diagonal matrix defined as $M_\ell = \text{diag}(m_\ell)$, we now define an index set that will

be used to recall which neurons will be re-initialized. The *survivor index set* is

$$\mathcal{S}_\ell \;=\; \{\, i \in \{1, \ldots, n_\ell\} \mid m_\ell(i) = 1 \,\}$$

**Pruning termination rule.** Let $W_\ell^{(r)} \in \{0,1\}^{n_\ell \times d_{\ell-1}}$ be the layer *mask matrix* at round $r$ (after applying mask $M_\ell^{(r+1)}$). A row $i$ is alive only if $\sum_j W_\ell^{(r)}[i,j] > 0$. The alive fraction is

$$\rho_\ell^{(r)} = \frac{1}{n_\ell} \sum_{i=1}^{n_\ell} \mathbb{I}\left( \sum_j W_\ell^{(r)}[i,j] > 0 \right), \qquad \rho_{\min}^{(r)} = \min\left(\rho_1^{(r)}, \rho_2^{(r)}\right), \tag{4.9}$$

where $\mathbb{I}(\cdot)$ is the *indicator function*, defined as

$$\mathbb{I}(P) = \begin{cases} 1, & \text{if the predicate } P \text{ is true} \\ 0, & \text{otherwise} \end{cases}$$

Pruning terminates once the smallest layer density is reached:

$$\rho_{\min}^{(r)} \le 1 - \zeta.$$

**Forward update masks.** Let the input (state) be $s \in \mathbb{R}^{d_s}$, hidden sizes/features $[n_1, n_2]$, and output (action) $a \in \mathbb{R}^{d_a}$. Nonlinearities are denoted by $\phi_1, \phi_2$ (e.g., ReLU/Tanh), the output squashing by $\psi$ (e.g., tanh), and $sc_a$ is the action-scaling term. With masks applied, the network forward pass is:

$$z_1 = \tilde{W}_1 x + \tilde{b}_1, \qquad h_1 = \phi_1(z_1), \tag{4.10a}$$

$$z_2 = \tilde{W}_2 h_1 + \tilde{b}_2, \qquad h_2 = \phi_2(z_2), \tag{4.10b}$$

$$a = sc_a \cdot \psi(W_3 h_2 + b_3)\,. \tag{4.10c}$$

**Masked forward and optimizer hygiene.** During IMP, the masked forward pass uses $\tilde{W}_\ell$ and $\tilde{b}_\ell$ in place of dense parameters. Additionally, to avoid stale optimizer momentum, mask consistency is enforced between pruning rounds by applying gradient masking after each step:

$$\nabla_{W_\ell} \leftarrow M_\ell \odot \nabla_{W_\ell},$$
$$\nabla_{b_\ell} \leftarrow m_\ell \odot \nabla_{b_\ell}.$$

This is implemented by resetting the optimizer after each prune and re-initializing it on the masked network using its pruned parameters, ensuring that optimization remains consistent with the active ticket structure.

### Initializing and Training Ticket

We now construct the smaller ticket using the weights from the late-rewind epoch, which were cached earlier, and the survivor index sets. Let $\mathcal{S}_1, \mathcal{S}_2$ be the survivor sets from the final pruning pass, and $n_1' = |\mathcal{S}_1|,\ n_2' = |\mathcal{S}_2|$. Define the *row-selection* matrices

$$R_1 \in \{0,1\}^{n_1' \times n_1}, \quad R_2 \in \{0,1\}^{n_2' \times n_2},$$

whose rows are standard basis vectors $v_i^\top$ for $i \in \mathcal{S}_1$ (resp. $\mathcal{S}_2$). Then the rebuilt (smaller) actor, instantiated with features $[n_1', n_2']$, copies *initial* parameters as

$$\boxed{\begin{aligned} W_1' &= R_1\, W_1^{(1)}, & b_1' &= R_1\, b_1^{(1)}, \\ W_2' &= R_2\, W_2^{(1)}\, R_1^\top, & b_2' &= R_2\, b_2^{(1)}, \\ W_3' &= W_3^{(1)}\, R_2^\top, & b_3' &= b_3^{(1)}. \end{aligned}}$$

This exactly matches the slicing:

$$W_1' \leftarrow W_1^{(1)}[\mathcal{S}_1, :], \quad b_1' \leftarrow b_1^{(1)}[\mathcal{S}_1], \tag{4.11a}$$

$$W_2' \leftarrow W_2^{(1)}[\mathcal{S}_2, :]\,[:, \mathcal{S}_1], \quad b_2' \leftarrow b_2^{(1)}[\mathcal{S}_2], \tag{4.11b}$$

$$W_3' \leftarrow W_3^{(1)}[:, \mathcal{S}_2], \quad b_3' \leftarrow b_3^{(1)}. \tag{4.11c}$$

**Remark.** The matrices $R_\ell \in \{0,1\}^{n_\ell' \times n_\ell}$ act as row-selection operators that project the original parameter space onto the surviving subspace of neurons. Each $R_\ell$ consists of standard basis vectors indexed by the survivor set $\mathcal{S}_\ell$, so $R_\ell$ and $R_\ell^\top$ are equivalent to index-based slicing operations in implementation.

After initializing the ticket network, it is trained on the training loop described in Algorithm 4.4(line 32).

To create a benchmark agent, a short network of the same size is randomly initialized and trained using Algorithm 4.5, which does not include the Lottery Ticket Hypothesis re-initializations.

**Algorithm 4.4** Ticket Agent Training with Iterative Magnitude Pruning

**Input:** Task, (target sparsity) $\zeta \in (0, 1)$, (Pruning Rounds) $R$, (epochs in-between prunes) $(K)$, Algorithm Parameters $(\kappa)$, Training Parameters $(\mathcal{T})$, $[n_1, n_2]$ Hidden Layer Features (`n_features`), Best Agent$(\Omega^B_{Full})$, Early(check-point) Agent$(\Omega^1_{Full})$ .

**Output:** Trained Short ticket agent, best metrics.

▷ *Initializations:* ◁

$\mathcal{M} \leftarrow$ MDP $\leftarrow (\mathcal{E})$

$\Omega(\boldsymbol{\pi}, \boldsymbol{Q}) \leftarrow$ Agent $\leftarrow (\mathcal{M}, \texttt{n\_features}, \kappa)$

$\mathcal{M}(S) \leftarrow \mathcal{P}_{minmax}(\mathcal{M}(S))$

▷ *minmax-preprocessor $\mathcal{P}_{minmax}$ normalizes observation states $S$ of the environment*

Compute the metrics for initialized agent (**J, R**)

$J \leftarrow \texttt{mean}(\textsc{ComputeMetrics}(\Omega_{\mathcal{D}=\{s_i,a_i,r_i,s'_i\}_{i=1}^N}, \mathcal{E}_\gamma))$

$R \leftarrow \texttt{mean}(\textsc{ComputeMetrics}(\Omega_{\mathcal{D}=\{s_i,a_i,r_i,s'_i\}_{i=1}^N}, 1.0))$

▷ *mean discounted reward & reward ref. Algorithm 4.2*

$\Omega'_B \leftarrow \Omega^B_{Full}$. ▷ *Load Best Pre-Trained agent reference*

$\Omega'^{\,0} \leftarrow \Omega^1_{Full}$. ▷ *Instantiate an agent with late-rewinded agent logged at initial epochs.*

$\theta^{(0)} \leftarrow \Omega'^{\,0}\{\theta'\}$ ▷ *Store layer parameters $W^{(1)}_\ell, b^{(1)}_\ell$ of the Initialized agent for all layers*

▷ *Iterative Magnitude Pruning Loop* ◁

Calculate $q = 1 - (1 - \zeta)^{1/R}$ per round fraction required to be pruned

**for** $r = 1$ to $R$ **do**:

  **for** $k = 1$ to $K$ **do**: ▷ *Train in between prunes for small K rounds*:

    Train $\Omega'^{\,i+1}_B \leftarrow \textsc{Learn}(\mathcal{M}, \mathcal{E}, \mathcal{T}, \Omega'^{\,i}_B)$ ▷ *Learn uses Algorithm Algorithm 4.3*

    Compute the metrics for agent (**J, R**)

  **for** $l$ in $[h1, h2]$ **do**: ▷ *Repeat for all hidden layers*:

    Load Layer Masks $m^{(k)}_\ell$ for the hidden layer. ▷ *Initialize mask for first run.*

    Calculate the combined bias and $L2$-norm score for each neuron.

    Zero out $q$ fraction neurons from the mask

    Store survivor indices$(\mathcal{S}_l)$

    Update and apply Mask $m^{(k)}_\ell$

  Reset optimizers and gradients ▷ *avoid stale momentum on zeroed rows*

  Calculate $\rho^{(r)}_\ell$ layer-wise % of neuron alive and $\rho^{(r)}_{\min}$ ▷ *Ref. to (4.9)*

  **if** $\rho^{(r)}_{\min} \leq \zeta$ **then**:

    **break**

Calculate $\texttt{n\_features}^{ticket} = [n'_1, n'_2]$ ticket layer number of features ▷ *Where,* $n'_l = |\mathcal{S}_l|$

$\theta^{ticket} \leftarrow \theta^{(0)} \odot \mathcal{S}$ ▷ *Calculate* $\theta^{ticket}$ *using survivor indices* ($\mathcal{S}_l$) *and initial parameters* ($\theta^{(0)}$) *ref.* (4.11)

$\Omega'^{\,0}_{random}(\boldsymbol{\pi}, \boldsymbol{Q}) \leftarrow \{\mathcal{M}, \texttt{n\_features}^{ticket}, \kappa\}$. ▷ *Initialize a random ticket agent*

$\Omega'_{ticket} \leftarrow \Omega_{random}(\theta^{\,random} \leftarrow \theta^{ticket})$ ▷ *Overwrite and Assign the initial states for the re-initializing trick for the ticket*

Compute the metrics for initialized ticket agent (**J, R**)
$J \leftarrow \texttt{mean}(\textsc{ComputeMetrics}(\Omega_{\mathcal{D}=\{s_i,a_i,r_i,s'_i\}_{i=1}^{N}}, \mathcal{E}_\gamma))$
$R \leftarrow \texttt{mean}(\textsc{ComputeMetrics}(\Omega_{\mathcal{D}=\{s_i,a_i,r_i,s'_i\}_{i=1}^{N}}, 1.0))$

▷ *Ticket Training Loop* ◁

Update $\text{best}_{\{\mathbf{J},\,\mathbf{R}\}} \leftarrow$(**J, R**)

**for** *each epoch* $i$ **do**

  Train $\Omega'^{\,i+1}_{ticket} \leftarrow \textsc{Learn}(\mathcal{M}, \mathcal{E}, \mathcal{T}, \Omega'^{\,i}_{ticket})$ ▷ *Learn uses Algorithm 4.3*

  Compute the metrics for initialized ticket agent (**J, R**)

  **if** $\mathbf{J} > \text{best}_{\{\mathbf{J}\}}$ **then**:

    Update $\text{best}_{\{\mathbf{J},\,\mathbf{R}\}} \leftarrow$(**J, R**)

    **save** best ticket agent($\Omega'^{\,i}_{ticket}$) (*at epoch=*$i$)

  **return** metrics **J, R**

**Algorithm 4.5** Baseline–Short Training (DDPG)

**Input:**

Task, (target sparsity) $\zeta \in (0, 1)$, Algorithm Parameters ($\kappa$),
Training Parameters ($\mathcal{T}$), $[n_1, n_2]$ Hidden Layer Features (`n_features`).

**Output:**

Trained random short agent, best metrics.

▷ *Initializations:* ◁

`n_features`$^{short} = [n_1', n_2'] = (1 - \zeta) \cdot [n_1, n_2]$

▷ *Scale* ***`n_features`*** *to match the ticket agent shape.*

$\mathcal{M} \leftarrow$ MDP $\leftarrow (\mathcal{E})$

$\Omega^{RandShortAgent}(\boldsymbol{\pi}, \boldsymbol{Q}) \leftarrow (\mathcal{M}, \texttt{n\_features}^{short}, \kappa)$

$\mathcal{M}(S) \leftarrow \mathcal{P}_{minmax}(\mathcal{M}(S))$

▷ *minmax-preprocessor* $\mathcal{P}_{minmax}$ *normalizes observation states* $S$ *of the environment*

Compute the metrics for initialized agent (**J, R**)

$J \leftarrow$ `mean`(ComputeMetrics($\Omega_{\mathcal{D}=\{s_i, a_i, r_i, s_i'\}_{i=1}^{N}}, \mathcal{E}_\gamma$))

$R \leftarrow$ `mean`(ComputeMetrics($\Omega_{\mathcal{D}=\{s_i, a_i, r_i, s_i'\}_{i=1}^{N}}, 1.0$))

▷ *mean discounted reward & reward ref. Algorithm 4.2*

Update best$_{\{\mathbf{J},\ \mathbf{R}\}} \leftarrow$(**J, R**)

▷ *Training loop* ◁

**for** *each epoch* $i$ **do**

  Train $\Omega^{i+1}_{RandTicketShort} \leftarrow$ Learn($\mathcal{M}, \mathcal{E}, \mathcal{T}, \Omega^{i}_{RandTicketShort}$)

  ▷ *Learn uses Algorithm 4.3*

  Compute the metrics for agent (**J, R**)

  **if** **J** $>$ best$_{\{\mathbf{J}\}}$ **then**:

    Update best$_{\{\mathbf{J},\ \mathbf{R}\}} \leftarrow$(**J, R**)

    **save** best agent($\Omega^{i}_{msh}$) (*at epoch=*$i$)

  **return** metrics **J, R**

## 4.4 Summary

This chapter consolidated the main contributions of this thesis and tied together the motivation and implementation details of the *Trikaya* framework. We proposed and stated the *central hypothesis*, that Lottery Ticket Hypothesis becomes more effective for high-dimensional control when learning occurs in environments that enforce geometric feasibility by restricting actions to the Null/Tangential space of a Constraint Space Manifold (e.g., ATACOM). This hypothesis provides the possible rationale for why combining constraint-manifold alignment with sparsification should produce stronger winning-ticket behaviour than pruning in unconstrained settings.

We then produced the mathematically expected advantages of learning lottery tickets on constraint manifolds by formalizing the relationship between the dimensionality of the tangent space and the effective policy capacity. Since feasible motion is restricted to a low-dimensional subspace, the actor need not represent arbitrary actions in the ambient space, and pruning is less likely to remove parameters critical to feasible control. As a result, the sparse ticket is naturally encouraged to allocate its remaining representational capacity along admissible directions, which explains the improved computational efficiency, memory reduction, and the tendency for constraint violations to remain low and stable under high sparsity.

Finally, we presented the practical contribution of this chapter, which consists of a complete and reproducible procedure for building and training *Trikaya* ticket networks using iterative magnitude pruning with late rewinding. The section describes how masks are constructed and updated, how survivor indices are extracted, how a smaller dense ticket is rebuilt using rewound initialization, and how training is performed with optimizer hygiene to maintain consistency across pruning rounds. We concluded by compiling implementation-ready algorithms that cover baseline full training, baseline short training, ticket training with Iterative Magnitude Pruning, and the metrics computation routine. These algorithms serve as a reference implementation for the experiments and benchmarking presented in the following chapters.

# Chapter 5

# Experimental Results and Performance Analysis

This chapter presents the performance of the ticket networks built with the *Trikaya* framework. It demonstrates its ability to train substantially smaller models by developing winning-ticket agents to accomplish the target tasks. We provide a comprehensive benchmarking of networks built with the *Trikaya* framework across multiple constrained control environments from ATACOM, to evaluate learning stability, convergence characteristics, and overall efficiency relative to dense baseline DDPG agents and their randomly pruned/shortened variants.

We begin by outlining the experimental setups and configurations in Section 5.1 and providing an overview of the training procedures. We then describe the use of separate training branches for analysis and benchmarking, along with the rationale for the chosen hyperparameters and Deep-RL algorithm. Next, we provide the evaluation setup for both constrained and standard MDP environments, which is crucial for supporting the Central Hypothesis 1.

In Section 5.2, we describe the performance analysis methodology. The environments are studied in increasing order of complexity to help the reader assess *Trikaya's* capabilities in a progressive manner. We start with the simplest environment, which is the *Circular Track* (Section 5.3). We then consider higher-dimensional planar tasks, which are *Planar Hit* (Section 5.4) and *Planar Defend* (Section 5.5), which share similar mechanics but differ in task objectives. Finally, we examine the *IIWA Hit* (Section 5.6), the most complex setting, featuring a 7-Degree Of Freedom (DOF) manipulator. Performance metrics are plotted to highlight learning characteristics. Additionally, simulation renderings are provided for each environment to connect data-driven results with the qualitative behaviour of the learned policies.

A separate study toggling the use of CSM-based learning is presented in Section 5.7 to further support Hypothesis 1. Two environments are considered to illustrate the contributions of CSM

towards the effectiveness of producing winning-ticket networks in the *Trikaya* Framework.

## 5.1 Training Setup

To rigorously analyze *Trikaya*, we focus on learning and convergence characteristics observed during training and evaluation. To clearly differentiate performance, three agent branches are trained and compared. Each branch serves a distinct purpose, offering a broader perspective on the key properties of interest. All training uses a single off-policy Deep-RL algorithm, namely DDPG. Evaluations span four environments of varying geometric and dynamic complexity, thereby improving the interpretability of results.

To further support the stated hypothesis (Hypothesis 1), which assumes that the Lottery Ticket Hypothesis benefits from learning within environments that utilize the Constraint Space Manifold, ATACOMs controlled switching between Constraint Space Manifold-enabled and disabled environments is employed. The effects of using ticket networks with manifolds are investigated independently, with benchmark results and corresponding analysis presented later in Section 5.7.

### 5.1.1 Evaluation Branching

The first baseline branch is a randomly initialized, full dense model trained on ATACOM environments. It reproduces the hyperparameters provided in the Appendix of [30] and is trained to saturate the rewards within 50–60 epochs. Models are evaluated by loading them separately to verify that learned actions reflect genuine solutions rather than reward hacking.

The second branch is the *(LTH) Ticket* agent obtained via the *Trikaya* framework. This is the principal benchmark. Section 4.3 provides details on constructing these agents. Ticket agents undergo substantial pruning/shrinking (often exceeding 95% parameter reduction) to show that compact models can still attain the intended performance when trained under *Trikaya*.

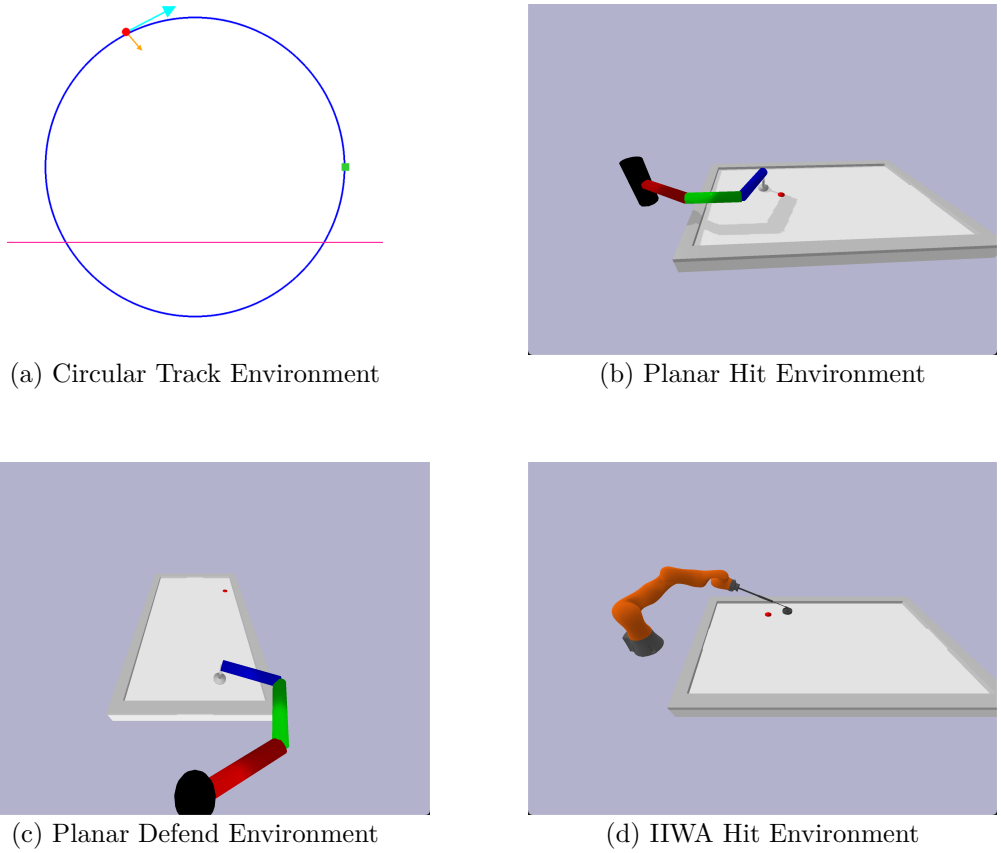
The third branch is a randomly initialized *shortened* model whose layer sizes match those of the ticket agent. It serves as a direct benchmark for the benefits of LTH-style initialization and masking by comparing against an equally small, randomly initialized network.

### 5.1.2 Experimental Environment Setups

We consider four ATACOM-enabled environments from Liu et al. [30]. As discussed earlier, ATACOM supports CMDP formulations that respect a Constraint Space Manifold, which is essential for applying *Trikaya* to train shortened models.

To improve the integrity of learned behaviours, each episode randomizes initial conditions (e.g., puck position, agent pose, approach angle). This discourages reward hacking via repeated trivial behaviours.

The environments are shown in Fig. 5.1. They are discussed with results in subsequent sections: Circular Track (Section 5.3), Planar-Hockey-Hit (Section 5.4), Planar-Hockey-Defend (Section 5.5), and IIWA-Hockey (Section 5.6).



(a) Circular Track Environment

(b) Planar Hit Environment

(c) Planar Defend Environment

(d) IIWA Hit Environment

Figure 5.1: The ATACOM environments used for the experimentation.

### 5.1.3 Paradigm and Parameter Justifications

We adopt DDPG rather than SAC primarily because it minimizes deviation from the original LTH literature and typically exhibits lower-variance policy execution dynamics [28]. Its simpler

actor–critic architecture, with only a single critic and target networks, aligns well with the Lottery Ticket Hypothesis objective of training sparse sub-networks under tight parameter budgets. In contrast, SAC includes dual critics, entropy tuning, and exploration regularization, which introduce extra variance and computational overhead when applied to sparse sub-networks. This is undesirable when the goal is to reduce the overall training cost. Nonetheless, SAC is still a strong and widely used algorithm for fully dense models, and for completeness Appendix A includes supporting results using SAC as the Deep-RL backbone.

The training pipeline parameters for all environments are consolidated in Tables 5.2 and 5.3 for reference and reproducibility. Table Table 5.1 details the agent structures for each environment. There is a noticeable increase in model parameters across environments. This growth reflects the necessary scaling of agent complexity to meet the demands of more challenging environments. These tables can be used to replicate the bulk of the training process.

Table 5.1: Hidden layer parameters for all of the models

| Environment | Full Agent | Ticket Agent | Short(Random) Agent | Layer Size reduction(%) |
|---|---|---|---|---|
| Circular Track | [100,100] | [1,1] | [1,1] | 99% |
| Planar Hockey Hit | [100,100] | [3,3] | [3,3] | 97% |
| Planar Hockey Defend | [100,100] | [5,5] | [5,5] | 95% |
| IIWA Air Hockey | [100,100] | [9,9] | [9,9] | 91% |

Model evaluation during training uses the built-in `evaluate` function in `mushroom_rl`. This executes the agent's policy for a specified number of steps/episodes (or from prescribed initial states), then resets the environment. The function returns a dataset $\mathcal{D}$ containing per-step tuples $(s, a, r, s_k, a_k)$ used to build the evaluation log. The log reports six metrics: $\mathbf{J}$, $\mathbf{R}$, $\mathbf{E}$, $\mathbf{C_{avg}}$, $\mathbf{C_{max}}$, and $\mathbf{C_{\dot{q}_{max}}}$. The entropy metric $\mathbf{E}$ is available only for algorithms whose policies expose it (e.g., SAC); it is absent for DDPG. In this chapter, we primarily track Discounted Return $\mathbf{J}$ and Maximum Constraint Violation $\mathbf{C_{max}}$ per epoch as the leading performance indicators, which provide the task performance and constraint adherence, respectively.

All experiments are run on an Apple MacBook Intel i5 processor (CPU). The average time per epoch and memory usage vary across branches. Consolidated time and memory summaries are provided in Tables 5.6, 5.8, 5.10 and 5.12.

### 5.1.4 Constrained and Unconstrained Environments

We also perform a benchmarking analysis in which the use of Constraint Space Manifold-aware environments is toggled on and off. In broad terms, this corresponds to comparing ATACOM-

Table 5.2: DDPG training and network hyperparameters used across the experimental branches.

| **Training Parameter** | Full Dense | Random Short | LTH (Trikaya) |
|---|---|---|---|
| Actor/Critic learning rate | $\{5e^{-3}, 1e^{-3}, 5e^{-4}\}$ | $\{5e^{-3}, 1e^{-3}, 5e^{-4}\}$ | $\{5e^{-3}, 1e^{-3}, 5e^{-4}\}$ |
| Epochs | 50 | 50 | 50 |
| Steps per epoch | 5000/10000 | 5000/10000 | 5000/10000 |
| Steps per fit | 1 | 1 | 1 |
| Episodes per test | 25 | 25 | 25 |
| Hidden layer sizes | $[100, 100]$ | $[100, 100]\cdot\zeta$ | $[100, 100]\cdot\zeta$ |
| Batch size | 64 | 64 | 64 |
| Replay warmup (init size) | 5000 | 5000 | 5000 |
| Replay capacity | 200000 | 200000 | 200000 |
| Soft update coefficient $\tau$ | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ |
| Discount factor $\gamma$ | 0.99 | 0.99 | 0.99 |
| Exploration noise $\sigma$ | 0.2 | 0.2 | 0.2 |
| Optimizer | Adam | Adam | Adam |

*Note:* $s$ denotes the sparsity. The actual hidden size can be computed after scalar multiplication.

enabled environments with environments that do not incorporate CSM, yielding constrained and unconstrained variants of the same task. This flexibility enables controlled experimentation and provides additional empirical evidence in support of Hypothesis 1.

As described above, training is conducted in two distinct environments: unconstrained and ATACOM-enabled settings. Two specific tasks are considered on the basis of clear interpretability: Circular Track Environment and Planar Hit Environment. Within each task, two model branches are deployed: a full-dense model, comprising dense layers with the maximum number of parameters (scaled upward to accommodate additional learnable elements in unconstrained environments), and a ticket model, which undergoes extensive shrinking and is then re-initialized and trained using LTH (with more lenient pruning scales compared to ATACOM-enabled settings).

The increased model size allocated to unconstrained scenarios directly affects the parameter counts in both the full and ticket models. Accordingly, pruning scale leniency is also applied to the ticket network in this analysis, since the environments are unconstrained. Table 5.4 details the hidden-layer sizes used.

The results of switching the use of Constraint Space Manifold on/off are further discussed in Section 5.7. These results help develop support around the stated Hypothesis 1.

Table 5.3: Physical and simulation parameters across ATACOM environments.

| **Sim. Params.** | Circular Track | Planar Hit | Planar Defend | IIWA Env. |
|---|---|---|---|---|
| Episode duration | 4 s | 2 s | 3 s | 3 s |
| Discount factor $\gamma$ | 0.99 | 0.99 | 0.99 | 0.99 |
| Sim. step size | 1/250 s | 1/250, 1/500 s | 1/250, 1/500 s | 1/500, 1/1000 s |

Table 5.4: Hidden layer parameters for the CSM toggle models

| Task | Environment Option | Full Agent | Ticket Agent |
|---|---|---|---|
| Circular Track | ATACOM (CSM enabled) | [100,100] | [4,4] |
| | Unconstrained (CSM disabled) | [200,200] | [20,20] |
| Planar Hockey Hit | ATACOM (CSM enabled) | [100,100] | [3,3] |
| | Unconstrained (CSM disabled) | [1024,1024] | [102,102] |

## 5.2 Performance Benchmarking of Trikaya

This section presents a structured evaluation of *Trikaya*. We examine how agent performance evolves as task dimensionality, constraint richness, and control complexity increase. Furthermore, by comparing performance under constrained and unconstrained settings, we aim to highlight the connection among ticket networks, constraint enforcement, and overall learning ability as stated in Hypothesis 1.

Each environment constitutes a distinct complexity tier. The evaluation begins with relatively simple planar control tasks and advances toward contact-driven robotic settings with complex dynamics. This progression highlights a gradual exploration of how the agent's learning ability is affected by increasing complexity. We further elaborate on the unique behaviours exhibited by the agents influenced by sparsity. For a fair and transparent comparison, all experiments follow a consistent training paradigm across branches (Tables 5.2 and 5.3). The three branches *Baseline-Full*, *Baseline-Short*, and *LTH-Trikaya* share identical hyperparameters, replay configurations, and random seeds, isolating the impact of LTH-based ticket training on CSM environments.

Performance is evaluated using metrics that capture both task performance and constraint satisfaction. The discounted return $\mathbf{J}$ and cumulative reward $\mathbf{R}$ quantify the overall control performance, while constraint-related measures, such as the average violation $\mathbf{C_{avg}}$ and maximum violation $\mathbf{C_{max}}$, depict the adherence to the constraints followed by each branch of the agent in different environments. We primarily track  and  per epoch as the primary performance indi-

cators, which measure task performance and constraint adherence, respectively. Supplementary indicators, like computational efficiency and random seed evaluation, are also analyzed to observe the actualization of the benefits produced using mathematical reasoning in Section 3.3.

The following subsections present results per environment. For each, we outline the setup, objectives, and constraints, followed by comparisons among the three branches. This progression from simple planar motion to multi-constraint robotic domains provides a comprehensive view of how *Trikaya* scales with complexity.

### 5.2.1 Visual Presentation of the Results

The performance results obtained from benchmarking different branches are presented as moving average plots and performance box plots, as explained below.

#### 5.2.1.1 Moving Average Plots

The performance curve plots report smoothed learning curves using a 10-epoch moving window. This reduces high-frequency noise and highlights sustained learning trends. This provides a reliable standard for comparing performance during training. Inherently, the 10-epoch mean reduces high-frequency noise and reveals the persistent direction of learning; any positive deviations reflect durable improvements to the signal rather than incidental variations. This is crucial as the claims made by this study are not about isolated spikes in the metrics and evaluation, but *a true sustained advantage across training windows*, which is precisely what the Moving Average Plots depict. The curves convey *trend stability* and *learning efficiency*. In particular, we focus on:

**Sample efficiency:** The rate at which a branch improves its moving-average reward indicates how effectively it uses training samples. Branches that achieve higher performance early on demonstrate superior optimization dynamics.

**Convergence stability:** The smoothness and variance of the curve appearing across later epochs tend to reflect how reliably the agent converges to its actual learned policy. The overall stable plateau indicates consistent policy refinement without chaotic behaviour.

By examining these properties together, the moving-average performance curves provide an in-depth understanding of the performance of different agent branches. To maintain transparency, we also overlay the raw data points, making the distribution of the outcomes visible.

It is worth noting that, in continuous control tasks, a higher smoothed mean implies that a policy is not merely lucky at a few steps but consistently produces higher returns in subsequent windows. This ensures that the improved performance is directly linked to learning the desired task for deployment, yielding stable, repeatable gains rather than occasional peaks.

#### 5.2.1.2 Performance Box Plots

Box plots are used to summarize the overall distribution and consistency of the learning outcomes across different branches. Unlike the moving-average performance curves, which focus on temporal evolution, the box plots summarize the *aggregate* behaviour of the trained policies across all evaluation episodes. This helps compactly represent the entire training spectrum as a meaningful distribution.

Box plots compactly characterize overall behaviour by the median, quartiles, and whiskers (the non-outlier range). Here, they confirm that the advantages observed in the smoothed curves persist across the full distribution of evaluation trials.

In reinforcement learning, such statistical visualizations are critical because returns often follow non-Gaussian and skewed distributions. The box plots, therefore, complement moving average curves by ensuring that claims of superior learning are supported by consistent dispersion. The subsequent sections examine the results for the environments shown in Fig. 5.1 in increasing order of complexity. Each section first outlines the environment mechanics, then presents and analyzes the results.

## 5.3 Circular Track Environment

We start our evaluation with the simplest environment, the Circular Track environment, depicted in Fig. 5.1(a). The agent's objective is to drive the "Red Point" as close as possible to the "Green Box" (Goal) positioned at (1,0), while adhering to the boundaries of the "Blue Circular Track" defined by the unit circle $x^2 + y^2 = 1$. The agent must also stay above the "Pink Height Line", a feature that increases the environmental complexity and encourages more sophisticated actions. The initial spawn location of the red spot varies randomly. The agent's control actions are the acceleration vector $a = [\ddot{x}\ \ddot{y}]^T$. The two arrows shown in Fig. 5.1(a) denote the x- and y-velocities in the point's local frame. Table 5.5 describes the mathematical formulation and constraints of the environment.

The moving-average performance plot (right) in Fig. 5.2 provides a temporal view of policy learning over 50 training epochs. Each curve is computed using a 10-epoch moving mean, which suppresses short-term oscillations and highlights sustained trends for each branch. The shaded region around each curve represents the empirical standard deviation across repeated trials, providing an estimate of variability. At the beginning of training, the LTH (blue) branch shows a noticeably faster rise in average return than both baselines, indicating superior *sample efficiency*. This early advantage persists, with the LTH curve stabilizing at a high-performance plateau well before epoch 20, exhibiting *convergence stability*. In contrast, the Baseline (Short) branch (orange) exhibits a nearly flat trajectory with minimal gains, indicating an undertrained, likely capacity-limited policy. The Baseline (Full) branch (green) eventually approaches a similar upper

Table 5.5: Constraint and reward formulation for the *Circular Track* environment.

| **Type** | **Mathematical Definition** | **Description** |
|---|---|---|
| *Equality Constraint* | $f(x, y): \; x^2 + y^2 - 1 = 0$ | Ensures the point moves strictly on the unit circle. |
| *Inequality Constraints* | $g_1(x, y): \; -y - 0.5 < 0,$<br>$g_2(\dot{x}): \; \lvert\dot{x}\rvert - 1 < 0,$<br>$g_3(\dot{y}): \; \lvert\dot{y}\rvert - 1 < 0$ | Enforces motion above $y > -0.5$ and bounds component-wise speeds. |
| *Reward Function* | $r(x, y) = \exp(-\sqrt{(x-1)^2 + (y-0)^2})$ | Encourages minimizing the Euclidean distance to the goal $(1, 0)$. |
| *Termination* | — | Episode terminates after a fixed horizon or upon numerical divergence. |

bound to LTH but requires significantly more epochs, consistent with dense models demanding longer optimization to reach comparable competence. The overlap of learning curves after sufficient training supports winning-ticket behaviour under fixed random seeds: if both branches learn similar *deterministic* policies on the same effective subset of parameters, their evaluation trajectories can coincide. This effect is more pronounced in this relatively simple task and becomes less pronounced as task complexity increases, with multiple near-optimal solutions emerging.

This moving-average performance plot communicates three key observations: (i) The LTH and Baseline (Full) branches occupy consistently higher bands, indicating stronger average performance across epochs; (ii) the Baseline (Short) branch fails to learn a competitive policy despite identical layer shapes to the LTH branch (differing only in mask and initialization); and (iii) after about 11 epochs, the Baseline (Full) and Ticket branches exhibit convergent reward updates. This likely arises because, under a fixed random seed, both agents stabilize on the same limited subset of effective weights preserved by the ticket branch, leading to identical initial evaluation states and subsequent policy behaviour. This pattern is indicative of winning-ticket behaviour in this network.

The performance box plot (on the left of Fig. 5.2) complements the temporal perspective by summarizing reward statistics across evaluation runs. Each box shows the Inter-Quartile Range

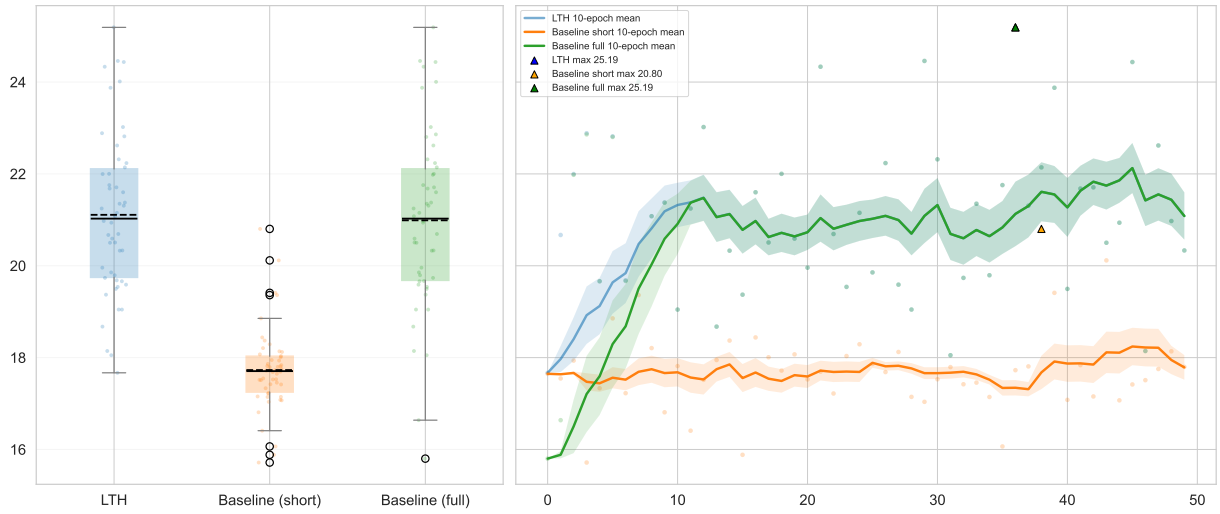
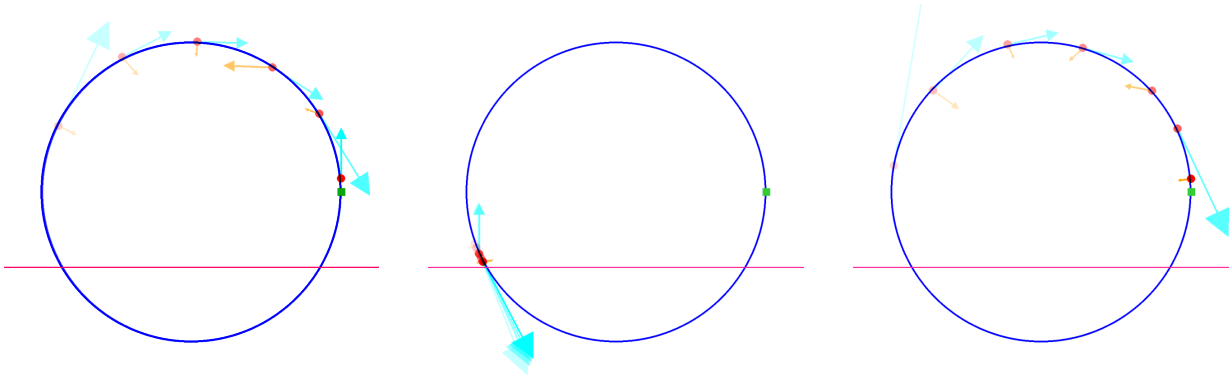



Figure 5.2: The training performance on Circular Track environment comparing the ticket agent trained with *Trikaya* framework (LTH) agent (Blue) against two baselines (Orange: random short training budget; Green: full budget). The horizontal axis shows training epochs, while the vertical axis conveys relative discounted return **J** magnitudes. (left) Distribution box plots, (right) rolling average curves.

(IQR) (middle 50%), with whiskers and points indicating variability and outliers. The box plot shows that the LTH branch closely tracks the full-baseline agent and achieves a median reward of approximately 21, significantly outperforming the random-short baseline (median $\approx$ 17.8). The LTH branch (blue) achieves a higher median and tighter interquartile range than the baselines, indicating strong, repeatable performance across seeds. The Baseline (Full) branch (green) attains a comparable median but with slightly higher variance; the Baseline (Short) branch (orange) remains low in median and compact yet under-performing in spread, confirming that restricted training prevents adequate policy formation.

Overall, these features suggest that training under the *Trikaya* framework yields faster, more stable control policies under identical evaluation conditions while substantially reducing training resources. A visual comparison is shown via simulations in Fig. 5.3. We can notice that the Ticket agent and the Full Baseline follow the track with comparable behaviour, while the Random Short branch, which has not matured into a meaningful policy, is unable to complete the loop. This highlights that the Ticket branch learns a coherent control strategy that the random-short agent fails to develop, demonstrating clear winning-ticket behaviour in this environment.

The constraint metric box plot in Fig. 5.4 summarizes the overall constraint violation statistics across the evaluation runs. From the visualization, the LTH branch (blue) exhibits a distribution similar to that of the full Baseline branch (green). Both have a lower and tighter IQR. The



(a) Full trained agent (b) Random short agent (c) Ticket agent

Figure 5.3: Animation results of the three branches for Circular Track environment.

narrow spread suggests stable, repeatable outcomes across random seeds.

Table 5.6 summarizes the computational resource footprint across the three training branches within the *Circular Track* environment, highlighting the computational efficiency achieved by the *Trikaya* framework. The *Total Training Time* denotes the start-to-end wall-clock duration of training. The *Avg. Time per Epoch* is the mean computation time required to complete a single training epoch. The *Avg. CPU Load* reports the mean processor utilization during training. In the simplest task, the LTH (Trikaya) agent and the Random Short baseline achieve nearly the same training-time reduction, reducing computation by roughly two-thirds compared to the Full model. However, LTH (Trikaya) retains this benefit *without sacrificing performance*. This indicates that the sparse ticket network retains sufficient functional capacity for control, while using *significantly fewer resources*.

Table 5.6: Resource comparison across the branches for *Circular Track* environment.

| Branch | Total Training Time | Average Time-per Epoch | Average Load-on CPU (%) | Relative Speed-up (%) | Layer-Size Reduction (%) |
|---|---|---|---|---|---|
| Baseline (Full) | 6 h 12 m | 7.4 min | 86% | (reference) | (reference) |
| Baseline (Short) | 2 h 05 m | 2.5 min | 78% | +197% | −99% |
| LTH (Trikaya) | 2 h 10 m | 2.6 min | 79% | +186% | −99% |

*Note:* All runs executed with simulation and debugging visualizer active.

In summary, the learning curves and box plots (Fig. 5.2) demonstrate the effectiveness of

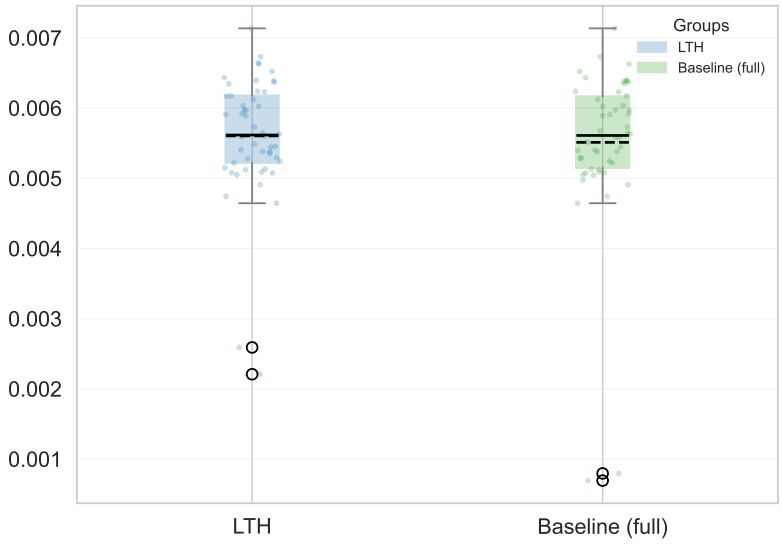



Figure 5.4: Box plot comparison of the maximum constraint violation metric ($c_{\max}$) between the *LTH* and *Baseline (full)* branches. A lower median and tighter interquartile range correspond to better constraint adherence and more stable satisfaction of the imposed equality and inequality limits. The visualization thus highlights that both branches maintain small yet similar constraint violation magnitudes throughout training.

*Trikaya* in this environment. The LTH-based agents reach high performance earlier and maintain it more consistently, while the box plots confirm that this improvement holds across the evaluation distribution. This confirms that the ticket agents have greater *sample efficiency* and *convergence stability*. Notably, pruning and structural reduction under LTH do not compromise representational capacity here; instead, they improve optimization stability, reduce variance, and match or exceed the performance of the fully dense model using substantially fewer parameters. A comparative visualization in Fig. 5.3 highlights the performance differences in achieving the target task. The results align with the conclusions drawn from the quantitative data. The ticket branch performs on par with the full baseline agent, whereas the random short branch struggles to develop a sufficiently adequate policy to accomplish the target task. To complement the quantitative evaluation presented in this section, simulation videos for all environments and training branches are provided in Table 5.13. These videos illustrate representative system trajectories and constraint interactions observed during policy execution.

Examining the constraint metric distribution (Fig. 5.4) alongside the computational resource profile (Table 5.6) further reinforces the advantages of the *Trikaya* framework in achieving both safe learning and computational efficiency. The table indicates that the ticket network completes training in significantly less time, demonstrating its superior resource efficiency compared to the

full baseline. At the same time, the constraint violations remain comparably low, matching the stability and safety of the full agent. These early results suggest that as the task complexity increases in subsequent environments, the benefits of the ticket network are expected to become more pronounced, producing not only convergence but also *improved adherence to safety constraints*.

The theoretical benefits discussed in Section 4.2 are anticipated to show up in more complex environments, which will be seen in the subsequent sections. This is primarily due to the simplicity of the circular track environment, which prevents a competitive edge from being apparent.

## 5.4 Planar Hit Environment

The Planar Hit environment, shown in Fig. 5.1(b), comprises a 3-joint planar robot whose end-effector operates on the tabletop to play air hockey in an attack configuration. In this setup, the puck is spawned at rest in a random location at the beginning of the episode. The primary objective is to strike the puck into the opponent's goal while keeping the robot's end-effector within the table boundaries. Table 5.7 summarizes the mathematical formulation and constraints of the environment. It is clear that this task is more complex than that of the Circular Track.

Fig. 5.5 presents reward distributions (left) and moving-average performance plot (right) across epochs for the three branches. The left panel illustrates the distribution of rewards as box plots, while the right panel shows the 10-epoch moving-average learning curves with shaded bands representing the empirical standard deviation across evaluation runs. The LTH agent achieves a median reward of approximately 40, substantially outperforming the random-short baseline (median $\approx 9$) and remaining close to the full baseline's performance. Both the LTH and full agents exceed a discounted return of 55 by the end of training, whereas the random-short branch plateaus near 20. Triangular markers denote maximum observed rewards, with the LTH configuration reaching roughly 77.2 compared to 35 for the random-short baseline, highlighting the efficiency and consistency of the pruned network.

Early training (right panel) shows the LTH (blue) branch rising faster than the random-short (orange) baseline, indicating better *sample efficiency*. After initial epochs, the Baseline (Full) (green) and LTH branches exhibit similar trajectories through mid-training-consistent with winning-ticket behavior-before diverging: the Baseline (Full) makes an abrupt jump in the mid-late range ($\sim$epoch 30), plausibly due to larger policy adjustments, while LTH approaches a similar upper bound smoothly, achieving comparable discounted returns with substantially fewer parameters.

Beyond convergence rate, the LTH curve exhibits lower fluctuation amplitude, suggesting more consistent gradient updates and reduced sensitivity to noisy rewards. Peak markers further indicate that LTH attains maximum returns similar to the full baseline and sustains them over

Table 5.7: Constraint and reward formulation for the *Planar Hit* environment.

| **Type** | **Mathematical Definition** | **Description** |
|---|---|---|
| *Inequality Constraints* | $g_1: -x_{ee} + x_{\text{table},l} < 0,$<br>$g_2: -y_{ee} + y_{\text{table},l} < 0,$<br>$g_3: y_{ee} - y_{\text{table},u} < 0,$<br>$g_{4,5,6}: q_i^2 - q_{i,l}^2 < 0,$<br>$g_{7,8,9}: \lvert\dot{q}_i\rvert - \dot{q}_{i,l} < 0$ | Define workspace, joint-limit, and velocity constraints that bound the end-effector and joints within physical limits. |
| *Reward Function* | $r = \begin{cases} \exp\left[-8\lVert p_{ee} - p_{\text{puck}}\rVert \text{clip}(\cos\theta, 0, 1)\right] - \lambda, \\ 1 + r_{\text{hit}} + 0.1 v_{x,\text{hit}} - \lambda, \\ 150. \end{cases}$ | The torque penalty $\lambda = 0.001\lVert a\rVert$. The reward case of $r$ in order is: The puck is not hit reward is based on closeness to the puck, The reward is based on the hit direction's closeness to goal and velocity of the puck, and the lastly if a goal is scored a high reward. |
| *Termination* | — | Episode ends when the puck leaves the table, a goal is scored, or the horizon is reached. |

a broader window. This highlights the smooth *convergence stability* of the LTH branch. These properties reflect sustained *training dynamics* in which sparsity propagates learning and improves stability. Despite having the same layer sizes as LTH, the random-short baseline underperforms throughout training, reinforcing that initialization/mask structure, not size alone, drives policy performance. Its nearly flat trajectory indicates limited policy maturation. Unlike the simpler Circular Track, overlapping trajectories are less pronounced here because multiple near-optimal solutions and contact dynamics produce diverse learning paths. Fixed random seeds align initial observations but cannot prevent divergent policy refinements in higher-complexity tasks.

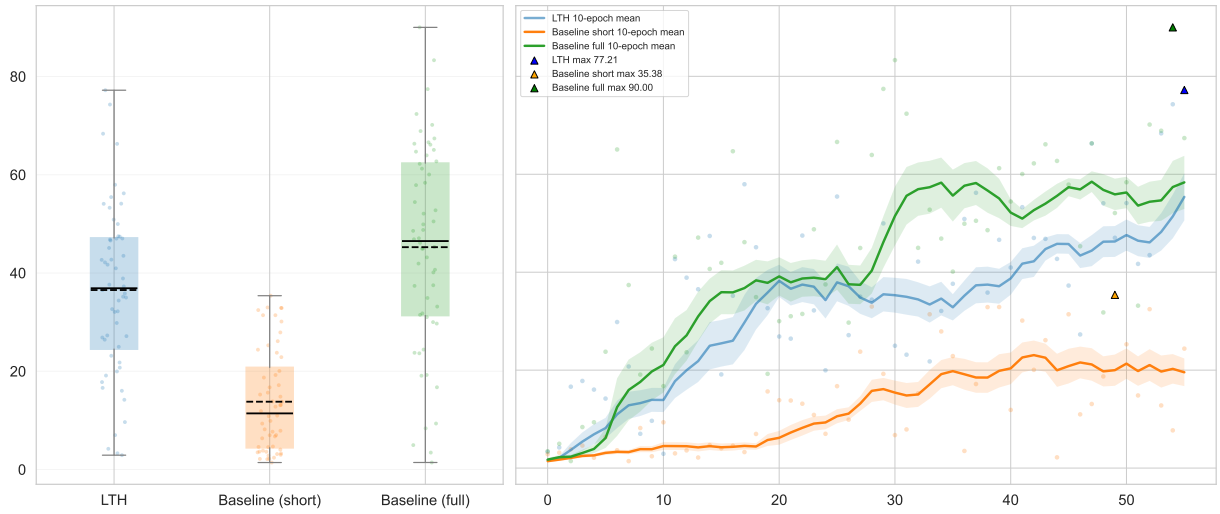



Figure 5.5: Training performance on Planar Hit comparing the ticket agent trained with the *Trikaya* framework (LTH; Blue) against two baselines (Orange: random short; Green: full). The $x$-axis shows training epochs, and the $y$-axis shows the relative discounted return $\mathbf{J}$. Left: reward distribution box plots. Right: 10-epoch moving-average learning curves with shaded empirical standard deviation.

The box plot in Fig. 5.5 (left) summarizes reward statistics across training runs. The LTH branch (blue) shows a higher median and tighter interquartile range than the random-short baseline (orange). Its median remains comparable to the Baseline (Full) (green), indicating successful learning of high-quality policies with repeatable outcomes across seeds. The random-short branch retains a low median and compact yet under-performing spread, confirming inadequate policy development under random initialization.

The constraint box plot in Fig. 5.6 summarizes $c_{\max}$ across evaluations. The LTH branch exhibits a lower, more compact distribution than the Baseline (Full), indicating more stable constraint satisfaction. This aligns with the analysis in Section 4.2, where sparser sub-networks are expected to reduce excursions along directions prone to constraint violations.

Table 5.8 summarizes computational resources for the three branches. The *Total Training Time* is wall-clock duration. The *Avg. Time per Epoch* is the mean epoch time. The *Avg. CPU Load* is the mean processor utilization. As task complexity increases to goal-directed planar contact motion, both LTH and random-short reduce training time relative to the Full network. However, only LTH preserves stable learning and policy quality, indicating that lottery-ticket initialization provides a reliable path to compact yet expressive policies under higher control complexity, while using *significantly fewer resources*.

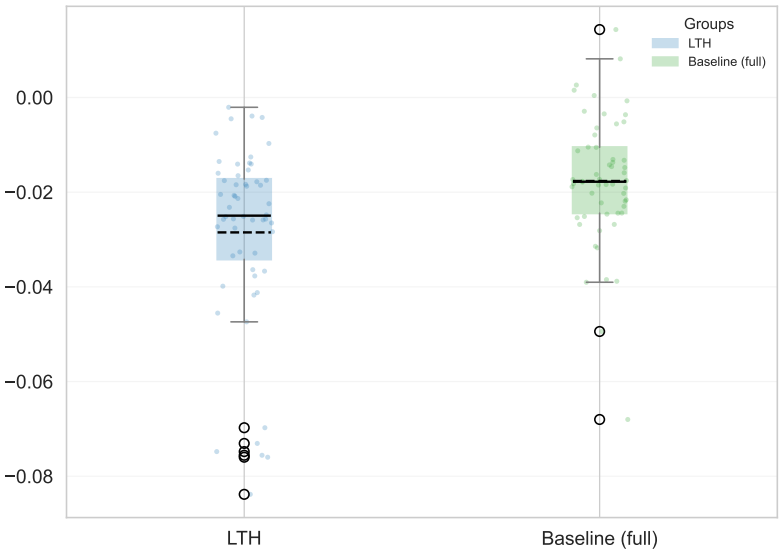



Figure 5.6: Constraint-violation metrics (box plots of $c_{\max}$) for Planar Hit.

In summary, the learning curves and box plots of Fig. 5.5 demonstrate the effectiveness of *Trikaya* in the Planar Hit environment. LTH-based agents reach high performance earlier and maintain it more consistently than the random-short baseline, while remaining close to the dense baseline's level. This again, confirms that the ticket agents have greater *sample efficiency* and *convergence stability*, even in high-dimensional planar tasks. Notably, pruning and structural reduction under LTH do not compromise representational capacity; rather, they improve optimization stability, reduce variance, and yield performance that matches the fully dense model while using substantially fewer parameters. To complement the quantitative evaluation presented in this section, simulation videos for all environments and training branches are provided in Ta-

Table 5.8: Resource comparison across branches for the *Planar Hit* environment.

| Branch | Total Training Time | Avg. Time per Epoch | Avg. CPU Load (%) | Relative Speed-up (vs. Full) | Model Size Reduction (%) |
|---|---|---|---|---|---|
| Baseline (Full) | 10 h 48 m | 12.9 min | 88% | (reference) | (reference) |
| Baseline (Short) | 3 h 45 m | 4.4 min | 82% | +190% | −97% |
| LTH (Trikaya) | 4 h 05 m | 4.8 min | 83% | +180% | −97% |

*Note:* Training executed with visualization enabled.

ble 5.13. These videos illustrate representative system trajectories and constraint interactions observed during policy execution.

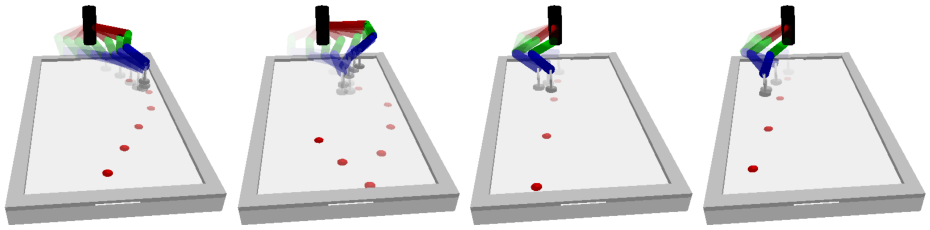
Examining the constraint-violation distribution of Fig. 5.6 alongside the computational resource profile in Table 5.8, further reinforces the benefits of *Trikaya* for both safe learning and efficiency. The LTH agent sustains lower and more compact $c_{\max}$ values while requiring significantly fewer computational resources than the full model. The table indicates that even with increased task complexity, the ticket network completes training here as well in substantially less time, demonstrating its superior resource efficiency compared to the full baseline and, at the same time, producing lower constraint violations, thereby improving the stability and safety of the full agent.

A visual comparison of all branches is shown as an animation in Fig. 5.7. From the figure, we observe that the Ticket agent and the Full Baseline perform on par, whereas the inadequately matured policy of the Random Short branch fails to accomplish the task. This further supports the conclusion that the Ticket branch successfully learns a functional policy that the random-short agent does not; thereby exhibiting clear winning-ticket behaviour.

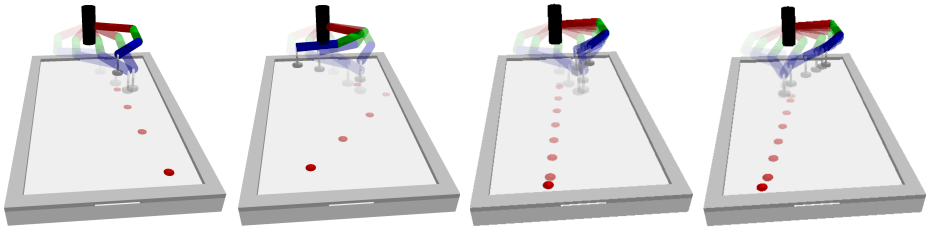
## 5.5 Planar Defend Environment

The Planar Defend environment, shown in Fig. 5.1(c), uses the same 3-joint planar robot used in Planar Hit environment, with its end-effector on the table to play air hockey in a defend scenario. The puck now moves dynamically with randomized velocity and initial position, requiring the agent to intercept and decelerate it while remaining within workspace boundaries. The primary objective is to defend against the puck and diminish its velocity, while the agent must remain within the table boundaries. Such a configuration introduces additional dynamic complexity and constraint coupling compared to the Planar Hit task, as described in Table 5.9.

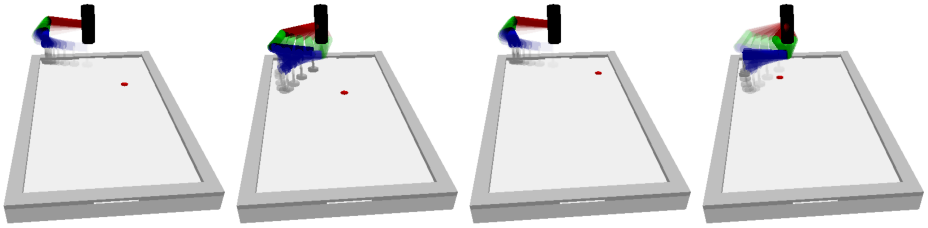
Fig. 5.8 presents the training performance results for the Planar Defend environment. At the beginning of training, the LTH (blue) branch exhibits a rapid rise in an average discounted return ($\mathbf{J}$) well before epoch 10. Thereafter, it steadily converges to a high performance for the remainder of the training budget. Triangular markers denote maximum recorded rewards, with the LTH configuration reaching approximately 74.68 a similar peak compared to the full-baseline's 79.99. The random short baseline is unable to get past 39.91. These results provide strong evidence that the ticket network trained via the *Trikaya* framework maintained its performance metrics, highlighting the ticket network's efficiency and consistency. This trajectory closely aligns with that of the Baseline (Full) branch (green), and both maintain upper reward bands indicating strong policy learning and stability. In contrast, the Random Short baseline (orange) again shows no indication of meaningful policy learning, reinforcing the critical role of structured initialization through LTH and pruning over mere network size reduction. The early and sustained high returns of the LTH agent highlight its superior *sample efficiency*.



(a) Full baseline agent multiple simulation runs.



(b) Ticket agent multiple simulation runs.



(c) Random Short agent multiple simulation runs.

Figure 5.7: Animation highlighting performance differences among branches in the *Planar Hit* environment.

The overlapping behaviour of the LTH and Baseline (Full) curves supports the presence of actual winning-ticket behaviour. However, unlike the near-perfect overlap observed in simpler environments, such as the Circular Track environment, the trajectories here remain similar but not entirely coincident, reflecting the increased complexity of the task. This divergence suggests that although both agents have discovered policies that learn similar underlying dynamics, their final parameterizations differ, leading to slight performance variations. The alternating dominance between the LTH and Full branches across training epochs further indicates that each has converged to a distinct, yet almost equally effective, local optimum. Overall, this reinforces that pruning under the *Trikaya* framework successfully preserves essential representational capacity, allowing the LTH agent to retain the core behavioural advantages of the full model despite operating with a substantially reduced parameter count.

The performance box plot, on the left of Fig. 5.8, complements the temporal view by summarizing episodic reward statistics across evaluation runs. The LTH branch achieves a median reward approaching 52, significantly surpassing the Random Short baseline (median $\approx 30$) and

Table 5.9: Constraint and reward formulation for the *Planar Defend* environment.

| **Type** | **Mathematical Definition** | **Description** |
|---|---|---|
| *Inequality Constraints* | $g_1: -x_{ee} + x_{\text{table},l} < 0,$<br>$g_2: -y_{ee} + y_{\text{table},l} < 0,$<br>$g_3: y_{ee} - y_{\text{table},u} < 0,$<br>$g_{4,5,6}: q_i^2 - q_{i,l}^2 < 0,$<br>$g_{7,8,9}: \lvert\dot{q}_i\rvert - \dot{q}_{i,l} < 0$ | Same workspace, joint, and velocity constraints as *Planar Hit*, ensuring feasible end-effector motion. |
| *Reward Function* | $r = \begin{cases} \exp[-3\lVert p_{\text{des}} - p_{ee}\rVert], \text{ puck not hit}, \\ 1 + \exp\left[-5\lvert x_{\text{puck}} + 0.6\rvert\right] + 5\exp\left[-5\lVert v_{\text{puck}}\rVert\right], \\ 0, \text{ short sides hit}, \\ -50, \text{ goal conceded}. \end{cases}$ | Rewards positioning near interception point and penalizes conceded goals; encourages quick, stable defensive behaviour. |
| *Termination* | — | Episode terminates when a goal or save occurs, or maximum steps are reached. |

closely matching the Full Baseline. This behaviour also suggests that, even under dynamically complex contact interactions, the LTH networks preserve robust control performance while exhibiting reduced training variance. The broader yet higher reward spread of the LTH branch indicates robust and consistent performance across random seeds, while the absence of outliers underscores policy stability and *stable convergence* rate. Meanwhile, the Random Short branch remains limited both in magnitude and variability, reaffirming its inability to learn a competent control policy.

As seen in Fig. 5.9, the LTH branch also validates the mathematical assumption discussed in Section 4.2 that predicts possible improvement on constraint violation metric, which can be attributed to smaller constraint violations due to their restricted representational capacity in directions that could violate constraints. These complementary insights demonstrate the suc-

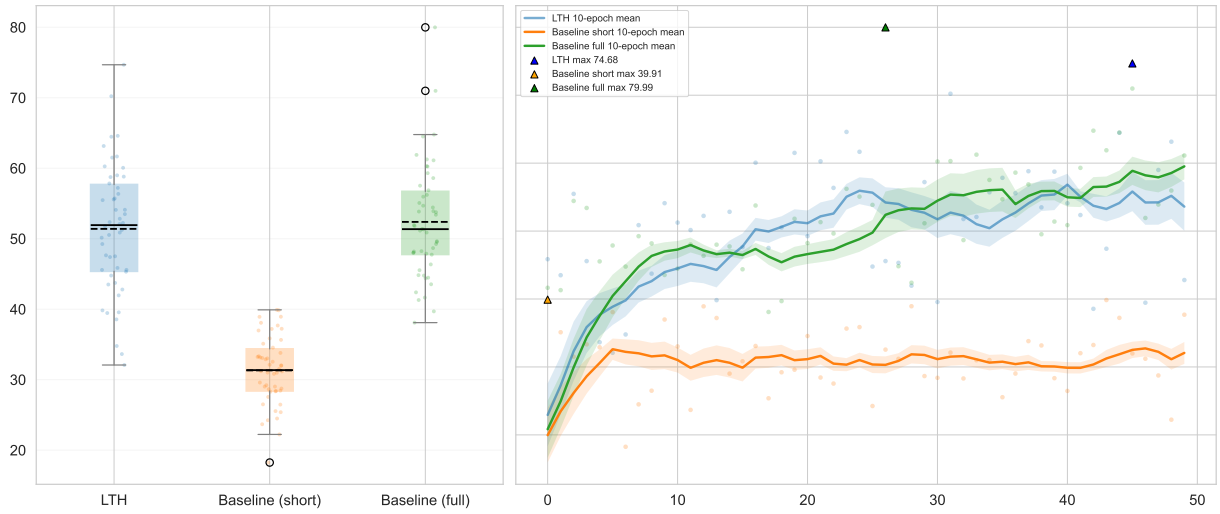



Figure 5.8: Training performance on the Planar Defend environment, comparing the ticket agent trained with the *Trikaya* framework (LTH; Blue) against two baselines (Orange: random short; Green: full). The $x$-axis shows training epochs, and the $y$-axis shows the relative discounted return $\mathbf{J}$. Left: reward distribution box plots. Right: 10-epoch moving-average learning curves with shaded empirical standard deviation.

cessful training of a shrunken network with greater stability and reduced variance, delivering performance on par with the fully dense model while using significantly fewer training resources.

Table 5.10 summarizes the computational resource footprint across the three training branches within the *Planar Defend* environment. This setting requires time-sensitive, continuous feedback-driven control to respond to incoming trajectories. The same relative patterns observed in the Planar Hit task persist here, with all branches requiring similar but slightly additional training time. Both reduced-size branches achieve substantially lower training times than the Full model. However, the LTH (Trikaya) agent maintains this efficiency while exhibiting stable optimization and policy consistency, whereas the Random Short baseline benefits only from reduced dimensionality. To complement the quantitative evaluation presented in this section, simulation videos for all environments and training branches are provided in Table 5.13. These videos illustrate representative system trajectories and constraint interactions observed during policy execution.

In summary, the learning curves and box plots (Fig. 5.8) demonstrate that *Trikaya*- trained ticket networks maintain performance characteristics similar to the dense baseline, while exhibiting greater consistency and a narrower and lower constraint spread. The Random Short baseline again fails to learn a functional policy, highlighting that network pruning guided by the LTH principle, not mere reduction in size, produces transferable, high-performing sub-networks.

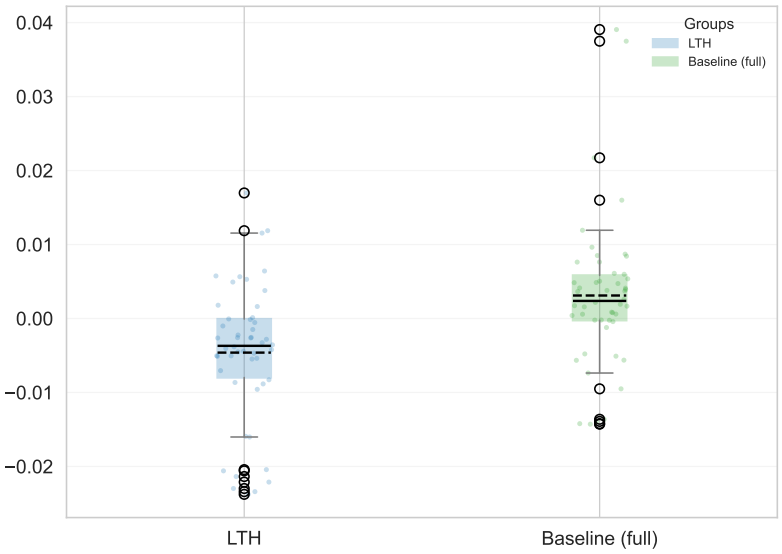



Figure 5.9: Resulting Constraint violation metrics (box plots of $c_{\max}$) for Planar Defend environment.

The constraint results (Fig. 5.9) confirm that *Trikaya* trained ticket sub-networks not only match full-model performance but also achieve safer, more stable behaviour through reduced constraint violation variance. Meanwhile, the computational profile (Table 5.10) mirrors that of the Planar Hit task, which can be attributed to the environment's similarity; this shows that LTH achieves equivalent performance with significantly fewer parameters and comparable training-time efficiency. These findings reinforce the hypothesis that sparse, structured sub-networks discovered via the *Trikaya* framework can replicate full-model performance under increasing dynamic complexity, in substantial alignment with the primary research hypothesis (Hypothesis 1).

A visual comparison of all branches is shown as an animation in Fig. 5.10. From the figure, we observe that the Ticket agent and the Full Baseline perform on par, whereas the inadequately matured policy of the Random Short branch fails to accomplish the task. This further supports the conclusion that the Ticket branch successfully learns a functional policy that the random-short agent does not; thereby exhibiting clear winning-ticket behaviour.

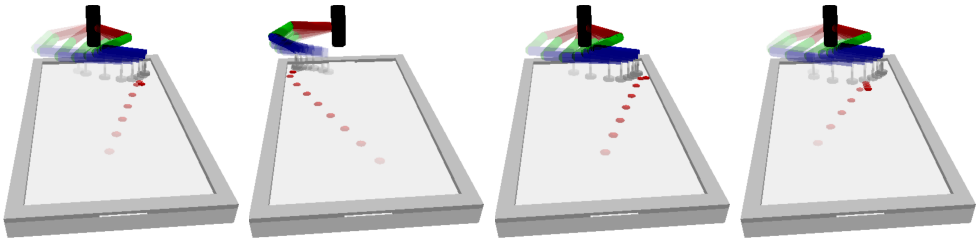
## 5.6 IIWA Hit Environment

The IIWA Hit environment, depicted in Fig. 5.1(d), uses a KUKA IIWA14, a 7-joint manipulator whose end-effector contacts the top of the table to play air hockey with a puck in an attack configuration. This is an extension of the earlier Planar Hit setup to a 7-DOF manipulator

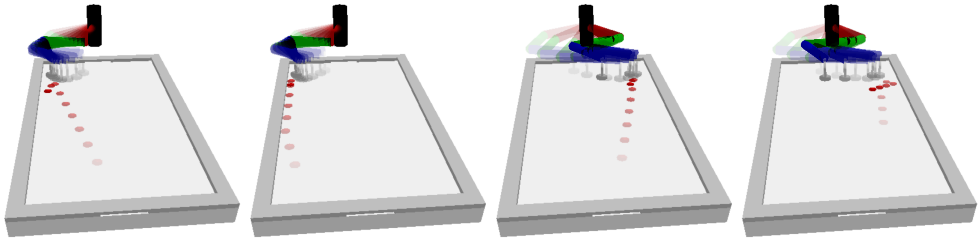
Table 5.10: Resource comparison across the branches for *Planar Defend* environment.

| Branch | Total Training Time | Avg. Time per Epoch | Avg. CPU Load (%) | Relative Speed-up (%) | Model Size Reduction (%) |
|---|---|---|---|---|---|
| Baseline (Full) | 11 h 45 m | 14.0 min | 85% | (reference) | (reference) |
| Baseline (Short) | 4 h 15 m | 4.9 min | 78% | +204% | −90% |
| LTH (Trikaya) | 4 h 30 m | 5.1 min | 79% | +196% | −90% |

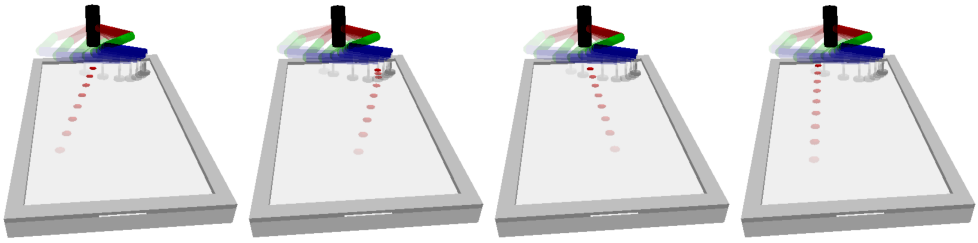
*Note:* CPU usage reflects concurrent logging and visualization.



(a) Full baseline agent multiple simulation runs.



(b) Ticket agent multiple simulation runs.



(c) Random Short agent multiple simulation runs.

Figure 5.10: Animation highlighting performance differences among branches in the *Planar Defend* environment.

performing goal-driven striking in three dimensions. The objective is to strike the puck into the opponent's goal while maintaining the end-effector's motion within feasible workspace limits,

subject to the additional constraint of keeping the end-effector on the table surface, as listed in Table 5.11. This configuration substantially increases the control dimensionality and introduces realistic contact dynamics, making it one of the most challenging tasks in the ATACOM collection.

Table 5.11: Constraint and reward formulation for the *IIWA Hit* environment.

| **Type** | **Mathematical Definition** | **Description** |
|---|---|---|
| *Equality Constraint* | $f: \; z_{ee} - z_{\text{table}} = 0$ | Keeps the mallet tip constrained to the planar table surface. |
| *Inequality Constraints* | $g_1: \; -z_4 + \hat{z}_{4,l} < 0,$<br>$g_2: \; -z_6 + \hat{z}_{6,l} < 0,$<br>$g_3: \; -x_{ee} + x_{\text{table},l} < 0,$<br>$g_4: \; -y_{ee} + y_{\text{table},l} < 0,$<br>$g_5: \; y_{ee} - y_{\text{table},u} < 0,$<br>$g_{6-11}: \; q_i^2 - q_{i,l}^2 < 0,$<br>$g_{12-17}: \; \lvert\dot{q}_i\rvert - \dot{q}_{i,l} < 0$ | Define workspace, height, joint, and velocity limits to ensure feasible 7-DoF motion during simulation. |
| *Reward Function* | $r = \begin{cases} \exp\left[-8\lVert p_{ee} - p_{\text{puck}}\rVert \text{clip}(\cos\theta, 0, 1)\right] - \lambda, \\ 1 + r_{\text{hit}} + 0.1 v_{x,\text{hit}} - \lambda, \\ 150. \end{cases}$ | Same structure as *Planar Hit*: encourages accurate strikes and scoring with minimal torque effort. |
| *Termination* | — | Episode ends when a goal is scored, the puck leaves the workspace, or the horizon is reached. |

Fig. 5.11 presents the training performance for the *IIWA Air-Hockey (Hit)* environment. The moving-average performance plot (right) illustrates the temporal evolution of learning over 50 training epochs, while the box plots (left) summarize reward distributions across evaluation trials. Each curve is computed using a running average to emphasize persistent learning trends and to reduce high-frequency oscillations. Shaded regions represent the empirical standard deviation obtained from repeated runs, providing an estimate of training variability.

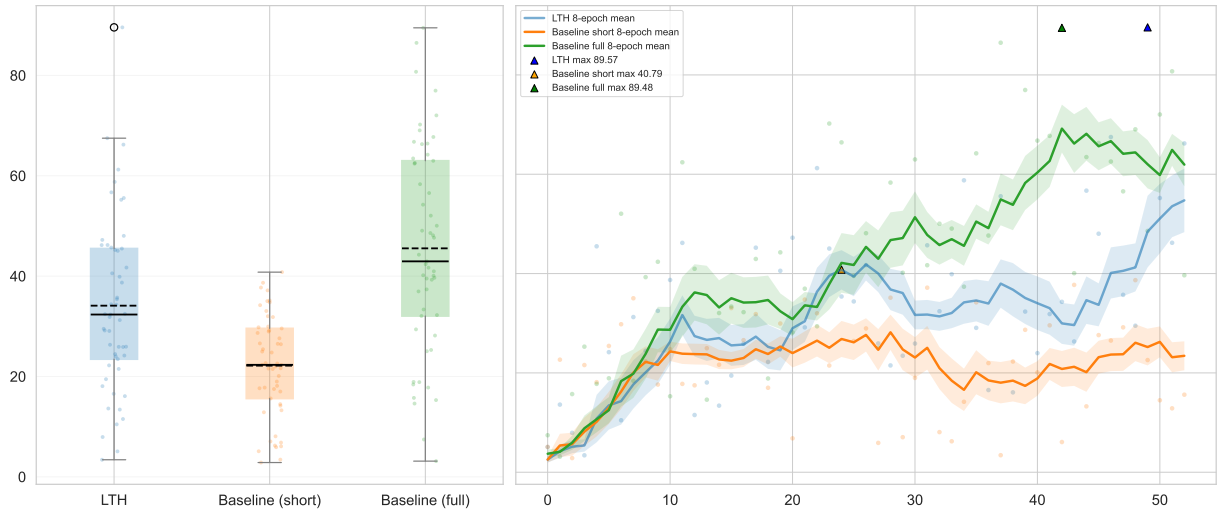



Figure 5.11: Training performance on the IIWA hit environment comparing the ticket agent trained with the *Trikaya* framework (LTH; Blue) against two baselines (Orange: random short; Green: full). The $x$-axis shows training epochs, and the $y$-axis shows the relative discounted return $\mathbf{J}$. Left: reward distribution box plots. Right: 10-epoch moving-average learning curves with shaded empirical standard deviation.

At the start of training, all branches exhibit a similar initial rise in average return. After approximately ten epochs, the Random Short baseline (orange) plateaus early and maintains its performance at that level throughout the remainder of training. In contrast, both the LTH (blue) and Full Baseline (green) branches continue to ascend toward higher reward regimes. The LTH branch begins to lag slightly behind the Full Baseline during the mid-phase of training. Eventually, it catches up, reaching a comparable high-reward plateau by the later epochs. This behaviour suggests that, in this more complex, high-dimensional setting, the pruned ticket network retains functional learning capability but *experiences less stable convergence* than the dense baseline. The maximum discounted returns achieved by the LTH and Full Baseline agents remain nearly identical (89.57 vs. 89.48), while the Random Short baseline peaks at only 40.79, underscoring its limited policy-learning capacity.

The performance box plot (left of Fig. 5.11) complements the temporal view by summarizing reward statistics across evaluation runs. The LTH branch exhibits a higher median and tighter distribution compared to the Random Short baseline (median $\approx 38$) and approaches the central tendency of the Full Baseline. The narrower interquartile range (IQR) for the LTH branch indicates reduced variance and improved generalization across evaluation seeds. However, the slightly lower upper quartile compared to the Full Baseline reflects the modest gap in convergence stability observed in the learning curves. Collectively, these results indicate that, while the

*Trikaya* trained sub-networks maintain high performance and parameter efficiency, the added dynamic complexity of the IIWA environment reveals minor limitations in stability and early-phase convergence relative to simpler planar tasks.

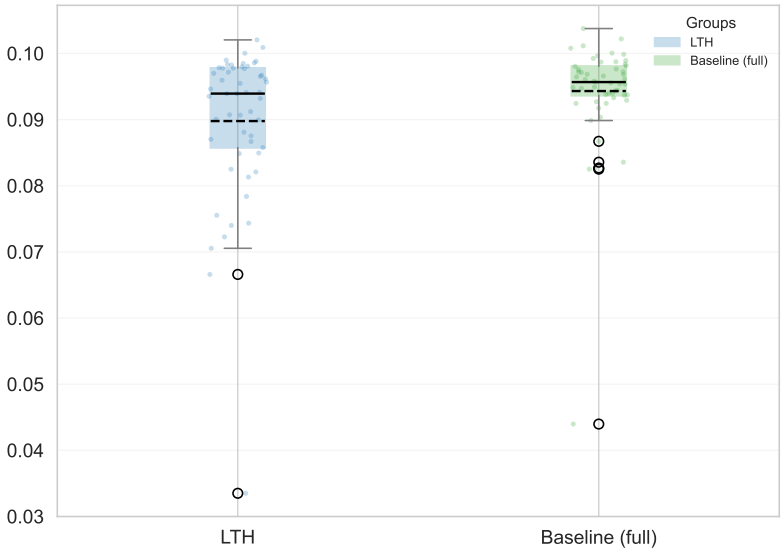


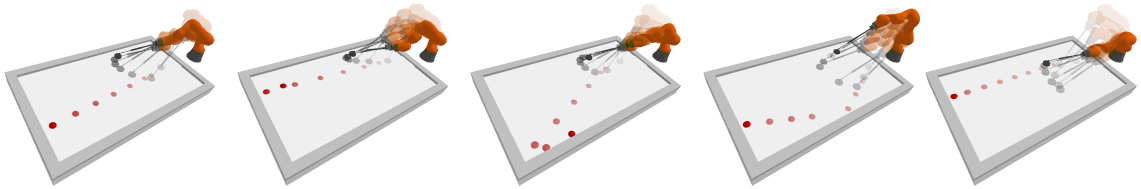

Figure 5.12: Resulting constraint violation metrics (box plots of $c_{\max}$) for the IIWA environment.

The constraint-violation comparison in Fig. 5.12 reinforces the benefits of using *Trikaya*. The LTH and Full Baseline branches maintain small overall violation magnitudes. However, the LTH branch exhibits a noticeably greater spread toward the lower quartile range of $c_{\max}$, indicating tighter adherence to the feasible motion manifold. This result implies that sparse networks trained under the *Trikaya* framework inherently bias policy updates toward constraint-compliant directions, leading to safer and more physically consistent learning behaviour. Such characteristics align with the theoretical expectations discussed in Section 4.2, where pruning is shown to reduce degrees of freedom associated with infeasible state-action trajectories.

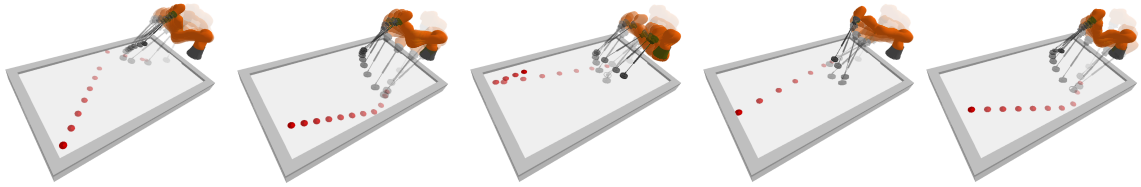
The computational resource profile for the IIWA environment mirrors that of the preceding planar tasks but scales proportionally with the task complexity. Table 5.12 summarizes these results, showing that all training runs require additional time compared to the Planar Defend setup due to the increased forward-pass cost and collision computations inherent to the 7-DoF manipulator. Despite this, the LTH (Trikaya) agent sustains a comparable reduction in overall training time and CPU utilization relative to the Full Baseline, achieving near-identical performance while utilizing roughly two-thirds less effective compute and a drastically smaller parameter budget. In contrast, the Random Short baseline maintains a similar computational footprint. Still, it fails

to learn a viable control policy, further underscoring that structured sparsity, rather than mere network reduction, is critical for efficient and functional policy learning.

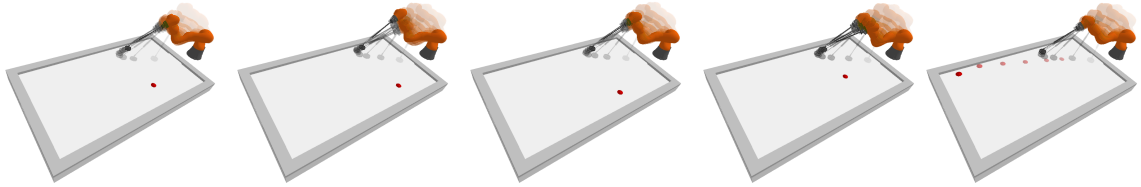
The behavioural difference among these agents can be visually observed in Fig. 5.13. The Random Short model repeatedly executes the same trajectory without any meaningful comprehension of the task objective, occasionally contacting the puck only when it happens to lie along this fixed motion path. This behaviour demonstrates reward hacking rather than genuine policy learning, as the agent optimizes a repetitive motion that superficially yields sparse rewards without achieving the true task goal. In contrast, both the LTH and Full Baseline agents display coherent, goal-directed motion patterns—tracking the puck, adapting their trajectories, and even successfully scoring goals in several runs. These qualitative observations reinforce the quantitative findings, illustrating that the *Trikaya* trained subnetworks acquire functional, interpretable policies rather than memorized action patterns.



(a) Full baseline agent multiple simulation runs.



(b) Ticket agent multiple simulation runs.



(c) Random Short agent multiple simulation runs.

Figure 5.13: Animation highlighting performance differences among branches in the *IIWA air-hockey hit* environment.

Table 5.12 compares the computational resource usage in the *IIWA Hit* environment, which involves higher kinematic complexity and a larger action space. Even under these increased demands, the LTH (Trikaya) agent achieves a substantial reduction in training time relative to the Full model. Importantly, it does so while maintaining stable learning dynamics, unlike the

Random Short branch, which benefits from reduced size but lacks the structured initialization provided by the ticket network. This demonstrates that the *Trikaya* sparsification strategy scales effectively to high-dimensional robotic control tasks. To complement the quantitative evaluation presented in this section, simulation videos for all environments and training branches are provided in Table 5.13. These videos illustrate representative system trajectories and constraint interactions observed during policy execution.

Table 5.12: Resource comparison across the branches for *IIWA Hit* environment.

| Branch | Total Training Time | Avg. Time per Epoch | Avg. CPU Load (%) | Relative Speed-up (%) | Model Size Reduction (%) |
|---|---|---|---|---|---|
| Baseline (Full) | 22 h 07 m | 26.4 min | 89% | (reference) | (reference) |
| Baseline (Short) | 7 h 30 m | 9.0 min | 82% | +192% | −91% |
| LTH (Trikaya) | 7 h 45 m | 9.2 min | 83% | +184% | −91% |

*Note:* Higher runtime reflects increased kinematic complexity.

In summary, the learning curves and reward distributions (Fig. 5.11) demonstrate that even in such a complex contact-rich control setting, subnetworks identified via the *Trikaya* framework retain performance comparable to the full model while using substantially fewer parameters. The LTH agent matches the dense model's returns, exhibits smoother convergence behaviour, and shows greater consistency across trials. Constraint-violation metrics (Fig. 5.12) confirm improved adherence to physical limits, while the computational profile (Table 5.12) highlights significant efficiency gains. Together, these findings extend the validation of the winning-ticket hypothesis into constrained, high-dimensional ATACOM tasks—strongly supporting the primary hypothesis (Hypothesis 1) that structured sparsity within CSM-enabled manifolds enhances both efficiency and stability of policy learning. The added complexity of the IIWA environment naturally introduces a minor performance gap and reduced convergence stability for the ticket networks, which remains consistent with expectations given the task's increased dimensional and dynamic difficulty.

## 5.7 A Comparative Analysis on Using CSM with LTH

Having established that we can form ticket networks performing on par with their fully dense counterparts, we now examine whether this ability fundamentally depends on the constrained-space formulation of ATACOM environments. Specifically, we aim to determine whether the discovery of winning tickets occurs only when the learning process is guided by the development of

policies in a CSM-enabled environment, thereby directly testing the primary research hypothesis (Hypothesis 1).

This analysis compares the performance of identical network architectures trained under two environmental settings: one with ATACOM's constraint manifold enabled and another with the constraint explicitly disabled. All other aspects, such as network initialization, Deep-RL algorithm, and pruning methods, remain constant. Additionally, an expanded computational budget is allocated to agents in the unconstrained environment to accommodate the additional learning requirement. This controlled setup isolates the role of environmental policy structure as the primary factor influencing ticket formation and policy convergence.

We expect that in unconstrained settings, sparse sub-networks fail to rediscover performing policies due to the lack of geometric constraint-guided parameter updates. Conversely, in CSM-enabled (ATACOM) environments, pruning should remain compatible with the manifold's tangent-space optimization, allowing ticket networks to retain representational fidelity and achieve baseline-level performance. The following sections present both cases to validate the primary Hypothesis 1.

### 5.7.1 Producing Tickets in CSM-Disabled Environments

ATACOM's constrained formulation embeds equality and inequality conditions directly into the optimization manifold, projecting policy gradients along feasible tangent directions and inherently bounding the learning space. When this structure is removed, however, learning proceeds in an unconstrained, high-dimensional action space with no geometric regularization. The resulting exploration becomes over-parameterized and noisy, relying solely on brute-force gradient descent without manifold-informed alignment. Under such conditions, sparse policies are expected to degrade rapidly, as the absence of constraint-oriented policy prevents pruned networks from preserving the functional subspaces necessary to maintain coherent control, Fig. 5.14 illustrates this outcome; Fig. 5.14(a) shows the Circular Track environment, where the LTH subnetwork (Green) fails to achieve meaningful learning despite convergence of the dense baseline (Blue), with rewards remaining near initialization and exhibiting high variance. Fig. 5.14(b) shows the Planar Defend environment, where the pruned subnetwork exhibits sporadic, unstable gains without establishing consistent convergence. In both cases, the sparse network exhibits minimal improvement over 60 epochs, with rolling averages remaining near their initialization values and a high variance in the reward distribution. Confirming the lottery ticket discovery is non-viable in the absence of manifold constraints, in sharp contrast to ATACOM environments (Fig. 5.15), providing supporting evidence for the Central Hypothesis 1.

In both the Circular Track and Planar Defend environments trained without CSM, the LTH ticket branches fail to recover meaningful policies. The moving-average curves for the sparse agents remain close to the dotted zero-return line throughout training, oscillating without upward

progression. In the box plots, the LTH distributions collapse near the lower quartile, indicating negligible reward accumulation and unstable learning. Despite extended training and equivalent compute budgets, these networks exhibit no sign of convergence and are effectively stuck near their initialization values.

The dense baselines, in contrast, converge smoothly to positive reward plateaus, highlighting that the underlying architecture and algorithm are sound; the breakdown arises purely from removing the constraints-aware policy formulation embedded in the environment. The ticket branches, deprived of manifold guidance, cannot maintain representational coherence. Their gradients diverge chaotically, and pruning amplifies this instability. This behaviour exemplifies the representational collapse predicted when sparse optimization proceeds without geometric constraints.

### 5.7.2 Producing Ticket in CSM-enabled Environment

When CSM is reintroduced through ATACOM's constrained dynamics, the learning behaviour changes dramatically. In these environments, physical feasibility constraints shape the optimization manifold, ensuring that policy updates occur only along valid tangential directions. This regularization reduces redundant parameter dependencies, making the learning process compatible with structured sparsity.

Fig. 5.15 demonstrates this successful lottery ticket discovery effect in ATACOM environments with manifold constraints. Fig. 5.15(a) shows the Circular Track environment, where ticket sub-networks (LTH: Blue) achieve performance parity with dense baselines (Green: full)) and converge to comparable asymptotic rewards near 250. The LTH network exhibits smooth learning dynamics with controlled variance, matching the baseline trajectory after epoch 30 despite utilizing significantly fewer parameters. Fig. 5.15(b) shows the Planar Defend environment, where lottery tickets not only match but occasionally exceed baseline performance, maintaining stable convergence throughout training. These results validate that constraint structured optimization landscapes preserve the representational capacity necessary for lottery ticket viability, supporting the central Hypothesis 1 that manifold geometry enables sparse network success in reinforcement learning

With CSM enabled in the Circular Track and Planar Defend environments, the LTH branches exhibit stable, sustained learning trajectories. The rolling-average curves for the ticket networks closely follow those of the dense baselines, converging to comparable asymptotic reward values despite having 90–95% fewer parameters. The box plots further show that the LTH distributions are centered near the baseline medians, with minimal variance and few outliers, indicating robust convergence.

The contrast with the CSM-disabled case is significant. The unconstrained environments showed oscillation and stagnation, while the CSM-enabled setups reveal smooth convergence

and high sample efficiency. The manifold-guided optimization ensures that pruning removes only redundant degrees of freedom while preserving the tangent-space representations essential for task completion. Consequently, the sparse tickets reproduce dense-agent performance, confirming that constraint-induced geometry provides the structural foundation necessary for viable lottery-ticket behaviour.

In summary, the comparative analysis clearly demonstrates that the structure of environmental constraints helps determine whether winning tickets can emerge. In non-ATACOM, unconstrained settings, sparse sub-networks fail to learn stable or meaningful policies even when granted generous computational budgets, confirming that sparsity alone is insufficient. Conversely, when training occurs under ATACOM's CSM-enabled formulation, the same pruning regime produces sub-networks that perform on par with dense agents, achieving stable convergence, high sample efficiency, and preserved constraint satisfaction.

The ATACOM advantage stems from constraint-induced policy regularization. Physical feasibility conditions naturally reduce the effective dimensionality of policy optimization, concentrating gradient information along task-relevant directions. Pruning is beneficial in this context, as it removes parameters that contribute minimally to the constrained optimization objective while retaining those aligned with the tangent manifold. The lottery ticket emerges not through random initialization luck but through systematic alignment with environmental structure. Unconstrained environments exhibit no such alignment. Networks depend on parameter redundancy to explore diverse gradient directions, many of which are irrelevant to optimal control. Pruning disrupts this redundancy-dependent exploration, causing a cascade of instabilities. The network cannot distinguish essential from auxiliary parameters without manifold constraints to guide the distinction. Consequently, aggressive sparsity degrades performance catastrophically in these settings.

This contrast validates the *Trikaya* framework's theoretical foundation and strongly supports the Central Hypothesis 1. The lottery ticket hypothesis holds when environmental dynamics impose geometric constraints on sparse representations.

## 5.8 Conclusion

The chapter presented a comprehensive experimental evaluation of the *Trikaya* framework across progressively complex ATACOM environments, ranging from simple planar motion to high-dimensional robotic manipulation. Through a series of controlled comparisons, we demonstrated that sparse ticket networks identified under the LTH principle can achieve performance comparable to their dense counterparts while requiring significantly fewer parameters and computational resources.

In the simpler tasks (Circular Track, Planar Hit and Planar Defend), the pruned sub-networks consistently matched the dense baselines in both learning stability and asymptotic performance,

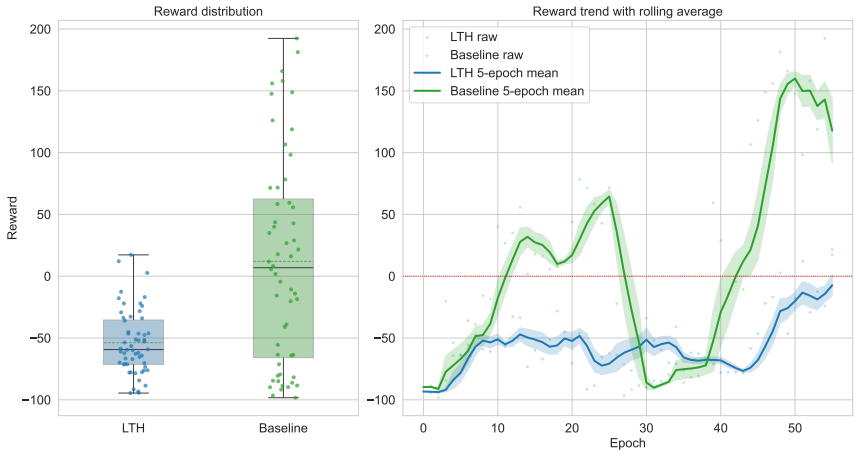
confirming the emergence of valid winning tickets. As task complexity increased (IIWA environment), the LTH branches maintained strong performance but displayed minor *convergence instability*, a natural consequence of the expanded control dimensionality. Across all cases, the results showed substantial gains in computational efficiency, sample utilization, and constraint satisfaction, validating the practical advantages of structured sparsity.

A comparative analysis between CSM-enabled (ATACOM) and unconstrained environments further revealed that geometric manifold constraints are essential for the successful discovery of winning tickets. While sparse networks trained without CSM collapsed to near-zero performance, those trained within ATACOM's constrained manifold achieved robust and stable policies. These findings collectively support the primary hypothesis (Hypothesis 1) that environmental structure and constraint-induced geometry play a critical role in enabling sparse, pruned networks to preserve representational capacity and learning efficacy.

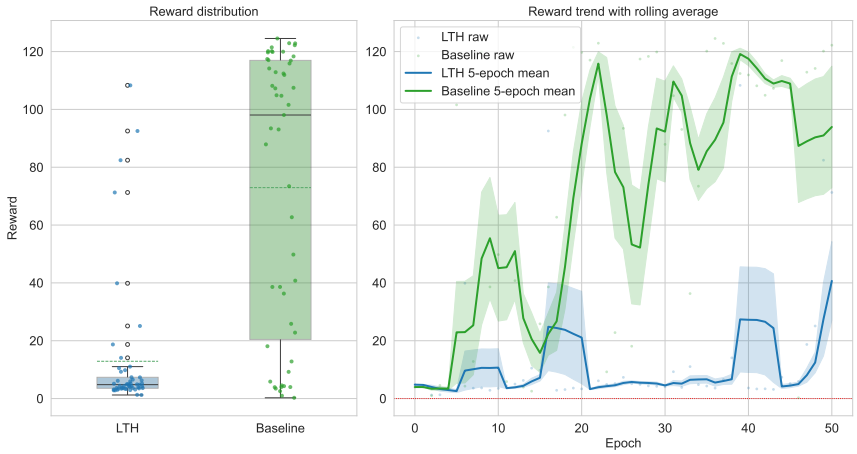
Overall, the experiments confirm that the *Trikaya* framework offers a strong pathway for training resource-efficient reinforcement learning agents that maintain performance parity with full models, thereby extending the lottery ticket hypothesis to constrained robotic control domains.

Table 5.13: Qualitative simulation result videos for all evaluated environments and training branches.

| Environment | Branch | Description | Video |
|---|---|---|---|
| Circular Track | Baseline Full | Dense policy executing smooth circular trajectories under simple kinematic constraints. | watch |
| | LTH (*Trikaya*) | Sparse policy retaining trajectory smoothness with reduced network capacity. | watch |
| | Random Short | Randomly pruned baseline illustrating sensitivity to unstructured sparsity. | watch |
| Planar Hit | Baseline Full | Dense policy learning contact-rich hitting motions within planar workspace constraints. | watch |
| | LTH (*Trikaya*) | Pruned policy preserving effective impact behaviour under constraint-aligned sparsity. | watch |
| | Random Short | Randomly pruned baseline exhibiting reduced consistency during striking motions. | watch |
| Planar Defend | Baseline Full | Dense policy learning collision-aware defensive trajectories. | watch |
| | LTH (*Trikaya*) | Sparse policy maintaining defensive performance under constraint-aware pruning. | watch |
| | Random Short | Randomly pruned baseline highlighting degraded defensive responsiveness. | watch |
| IIWA Air-Hockey | Baseline Full | Full-capacity policy executing high-speed striking motions under constraint enforcement. | watch |
| | LTH (*Trikaya*) | Sparse constraint-aware policy achieving stable and precise impact behaviour. | watch |
| | Random Short | Randomly pruned baseline demonstrating instability in high-speed control. | watch |

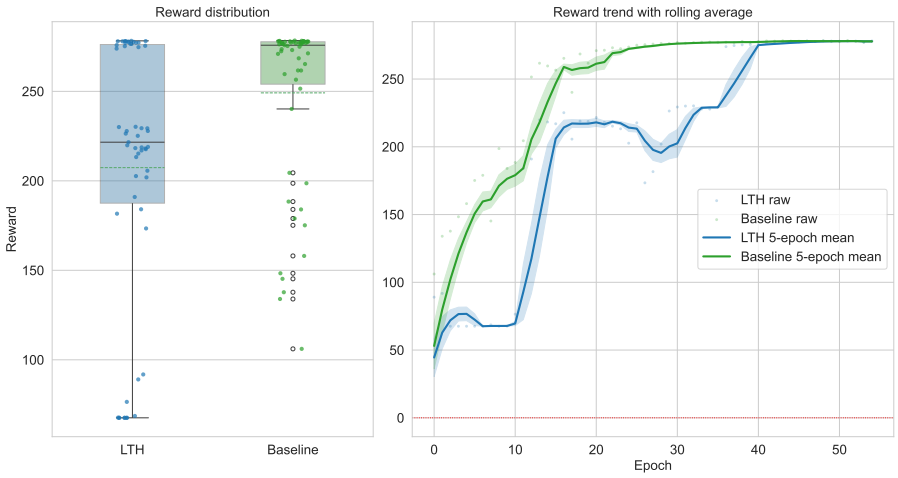



(a) Circular Track environment without CSM.

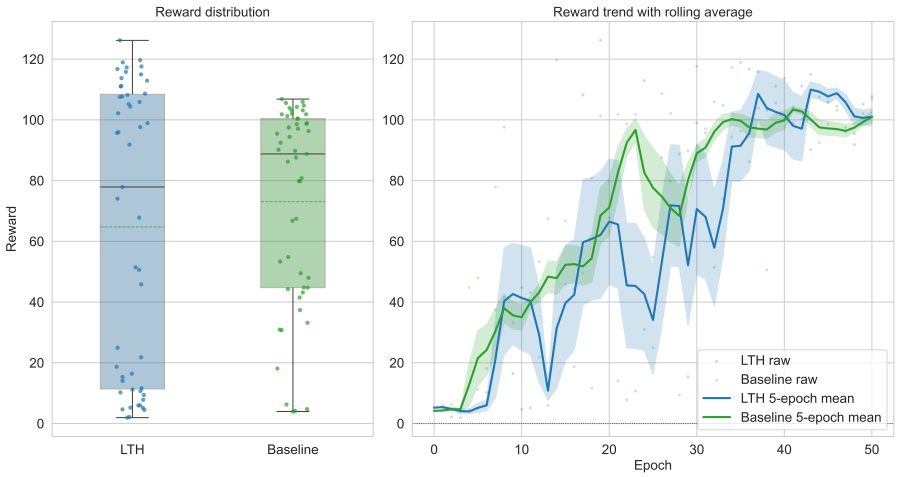



(b) Planar Defend environment without CSM.

Figure 5.14: Training performance in unconstrained environments, comparing the ticket agent trained with the *Trikaya* framework (LTH: Blue) against the baseline Green: full). The $x$-axis shows training epochs, and the $y$-axis shows the relative discounted return **J**. Left: reward distribution box plots. Right: 10-epoch moving-average learning curves with shaded empirical standard deviation.




(a) Circular Track environment with CSM.




(b) Planar Defend environment with CSM.

Figure 5.15: Training performance in ATACOM based environments with manifold constraints, comparing the ticket agent trained with the *Trikaya* framework (LTH; Blue) against the baseline Green: full). The $x$-axis shows training epochs, and the $y$-axis shows the relative discounted return $\mathbf{J}$. Left: reward distribution box plots. Right: 10-epoch moving-average learning curves with shaded empirical standard deviation.

# Chapter 6

# Conclusion

In this chapter, we provide some concluding remarks, and a brief summary of the work presented in this thesis. We will be revisiting several of the key high-level ideas introduced throughout this work. The reader is also presented with potential future aspects for extending this research, offering directions through which the proposed framework may be further explored, refined, and improved based on the findings observed in the experimental evaluations.

In this work, a hypothesis is proposed linking the increased efficiency of discovering winning tickets to the use of CSM based environments. Building upon this idea, a unified reinforcement learning framework which is termed *Trikaya* has been developed to enable the discovery and training of highly sparse winning-ticket sub-networks for robotic control tasks. By leveraging CSM environments that operate on tangential manifolds, the framework constrains learning to physically plausible action spaces. Integrating off-policy Deep-RL actor-critic methods with the structural advantages of the LTH, and the geometric constraint adherence enforced through ATACOM's CSM enabled environments; Trikaya allows compact policies to be trained reliably from scratch while ensuring that all learned actions remain feasible. The resulting sub-networks operate with as little as 5–10% of the original parameter count, yet preserve strong task-relevant representational capacity across robotic domains of varying complexity.

The effectiveness of the proposed methodology is demonstrated through comprehensive simulations on ATACOM environments of increasing complexity, which includes Circular Track, Planar Hit, Planar Defend, and the 7-DOF IIWA Air-Hockey tasks. Across these domains, the Trikaya sub-networks consistently match or closely approximate the performance of their dense, full-parameter counterparts while exhibiting smoother convergence behavior and reduced variance in evaluation returns. In several tasks, the sparse networks even outperform the full models in sample efficiency and constraint satisfaction. In contrast, when trained in non-CSM environments, pruned networks fail to learn meaningful policies, oscillating around zero return and demonstrating clear representational collapse. This stark performance gap confirms that

constraint-aligned learning landscapes are essential for increasing the emergence chances of winning tickets in reinforcement learning based tasks; this contrast provides ample of support to the proposed Hypothesis 1.

Computationally, the proposed sub-networks yield large efficiency gains. Training time is reduced by up to two-thirds compared to dense networks, CPU load is consistently lower, and the reduced parameter sizes significantly ease memory and deployment burdens. These savings make Trikaya particularly appealing for *low-resource robotic platforms*, where hardware constraints often prohibit the deployment of large deep neural policies. Despite the drastic reduction in capacity, the learned policies remain stable, safe, and capable of achieving high performance across complex, contact-rich tasks.

## 6.1 Limitations and Future Directions

Although this work presents strong empirical evidence for the viability of constraint-aligned winning tickets in robotic reinforcement learning, several important directions remain open for further investigation.

First, the theoretical underpinnings of why CSM-aligned environments appear to help sparse agents harmonize with gradient-based learning warrant deeper mathematical analysis, particularly in connection with manifold optimization and constraint-projected policy updates. A more formal characterization of how winning tickets emerge under action-space projection onto constraint manifolds would help clarify whether the observed behavior is a general property of constrained learning or specific to the environments studied here. In this context, developing theoretical criteria that relate sparsity, constraint satisfaction, and policy stability remains an important open problem.

Second, while the environments considered span multiple levels of difficulty, extending the *Trikaya* framework to real-world robotic systems would provide more conclusive validation. Addressing the sim-to-real gap requires careful consideration of sensor noise, actuation delays, and unmodeled dynamics, as well as an evaluation of whether the computational savings observed in simulation translate into practical advantages on embedded controllers. As an intermediate step, implementing ticket-based constrained policies within a Robot Operating System (ROS)-based framework would offer a more realistic and modular platform for bridging simulation and physical deployment.

Third, scaling *Trikaya* to higher-dimensional and more complex tasks, such as contact-rich manipulation, dual-arm systems, humanoid control, or multi-agent coordination, would help test the limits of sparsity and constraint geometry at larger scales. Such extensions would further clarify whether winning-ticket behavior persists as task dimensionality and interaction complexity increase.

Finally, future work may explore more adaptive pruning strategies that move beyond fixed magnitude-based schedules. Structured or constraint-aware pruning techniques that explicitly optimize the trade-off among return, constraint satisfaction, and computational cost could further improve policy stability and reduce training variance. For instance, pruning criteria that prioritize parameters critical to maintaining feasibility within the constraint manifold may yield more robust sparse policies.

## 6.2 Summary

Overall, the results of this thesis demonstrate that *Trikaya* provides a promising foundation for scalable, resource-efficient, and physically coherent reinforcement learning in robotics. By combining the representational economy of winning-ticket subnetworks with the geometric feasibility of constraint-manifold learning, this framework takes a step toward practical deployment of advanced RL controllers on low-power robotic platforms, while preserving the robustness and safety required for complex real-world control tasks.

# References


[1] Joshua Achiam et al. *Constrained Policy Optimization*. 2017. arXiv: 1705.10528 [cs.LG].

[2] Dmitry Berenson et al. "Manipulation Planning on Constraint Manifolds". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2009, pp. 625–632.

[3] Lukas Brunke et al. "Safe and Robust Robot Learning: A Survey". In: *Annual Review of Control, Robotics, and Autonomous Systems* 5 (2022), pp. 411–444.

[4] Tianyi Chen et al. "Only train once: A one-shot neural network training and pruning framework". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 19637–19651.

[5] Gal Dalal et al. *Safe Exploration in Continuous Action Spaces*. 2018. arXiv: 1801.08757 [cs.LG].

[6] Gabriel Dulac-Arnold, Daniel J. Mankowitz, and Todd Hester. "Challenges of Real-World Reinforcement Learning". In: *Machine Learning* 110.9 (2021), pp. 2419–2468.

[7] Utku Evci et al. "Rigging the Lottery: Making All Tickets Winners". In: *International Conference on Machine Learning (ICML)*. 2020.

[8] Tingxiang Fan et al. "Distributed Multi-Robot Collision Avoidance via Deep Reinforcement Learning for Navigation in Complex Scenarios". In: *International Journal of Robotics Research* 39.7 (2020), pp. 856–892.

[9] Jonathan Frankle and Michael Carbin. "The Lottery Ticket Hypothesis: Training Pruned Neural Networks". In: *CoRR* abs/1803.03635 (2018). arXiv: 1803.03635. URL: http://arxiv.org/abs/1803.03635.

[10] Jonathan Frankle et al. "Stabilizing the Lottery Ticket Hypothesis". In: *arXiv preprint* arXiv:1903.01611 (2020).

[11] Scott Fujimoto, Herke van Hoof, and David Meger. “Addressing Function Approximation Error in Actor-Critic Methods”. In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. 2018.

[12] Laura Graesser et al. “The State of Sparse Training in Deep Reinforcement Learning”. In: *Proceedings of the 39th International Conference on Machine Learning (ICML)*. Vol. 162. Proceedings of Machine Learning Research. 2022, pp. 7742–7765.

[13] Shixiang Gu et al. “Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 3389–3396.

[14] Tuomas Haarnoja et al. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”. In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. 2018.

[15] Tuomas Haarnoja et al. “Learning agile soccer skills for a bipedal robot with deep reinforcement learning”. In: *Science Robotics* 9.89 (2024), eadi8022.

[16] Kris Hauser. “Constraint-Based Motion Planning”. In: *IEEE Robotics & Automation Magazine* 16.3 (2009), pp. 22–33.

[17] Diego Ibarz et al. “How to Train Your Robot with Deep Reinforcement Learning: Lessons We Have Learned”. In: *Annual Review of Control, Robotics, and Autonomous Systems* (2021). Preprint version.

[18] Leonard Jaillet and Josep M. Porta. “Path Planning Under Kinematic Constraints by Rapidly Exploring Manifolds”. In: *IEEE Transactions on Robotics* 29.1 (2013), pp. 105–117.

[19] Mrinal Kalakrishnan et al. “STOMP: Stochastic Trajectory Optimization for Motion Planning”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, 2011, pp. 4569–4574.

[20] Juhwan Kim, Jaeheung Park, and Il Hong Suh. “Tangent Bundle RRT: A Randomized Algorithm for Constrained Motion Planning”. In: *Robotica* 34.1 (2016), pp. 202–225.

[21] Zachary Kingston, Mark Moll, and Lydia E. Kavraki. “Sampling-Based Methods for Motion Planning with Constraints”. In: *Annual Review of Control, Robotics, and Autonomous Systems* 1 (2018), pp. 159–185.

[22] Jens Kober, J. Andrew Bagnell, and Jan Peters. “Reinforcement Learning in Robotics: A Survey”. In: *International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274.

[23] Oliver Kroemer, Scott Niekum, and George Konidaris. “A Review of Robot Learning for Manipulation: Challenges, Representations, and Algorithms”. In: *Journal of Machine Learning Research* 22.30 (2021), pp. 1–82.

[24] Solomon Kullback and Richard A. Leibler. “On Information and Sufficiency”. In: *The Annals of Mathematical Statistics* 22.1 (1951), pp. 79–86.

[25] Rushil Kumar and Wail Gueaieb. “Leveraging Constraint Space Manifolds and Lottery Tickets to Learn Complex Control Tasks with Pruned Neural Networks”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. (To be submitted). Pittsburgh, Pennsylvania, USA, Sept. 2026, pp. 1–6.

[26] Sergey Levine et al. “End-to-End Training of Deep Visuomotor Policies”. In: *Journal of Machine Learning Research* 17.39 (2016), pp. 1–40.

[27] Sergey Levine et al. “Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection”. In: *International Journal of Robotics Research* 37.4–5 (2018), pp. 421–436.

[28] Timothy P. Lillicrap et al. “Continuous Control with Deep Reinforcement Learning”. In: *arXiv preprint* arXiv:1509.02971 (2015).

[29] Yating Lin, Glen Chou, and Dmitry Berenson. *Improving Out-of-Distribution Generalization of Learned Dynamics by Learning Pseudometrics and Constraint Manifolds*. 2024. arXiv: 2403.12245 [cs.RO]. URL: https://arxiv.org/abs/2403.12245.

[30] Puze Liu et al. “Robot Reinforcement Learning on the Constraint Manifold”. In: *5th Annual Conference on Robot Learning*. 2021. URL: https://openreview.net/forum?id=zwo1-MdMl1P.

[31] Puze Liu et al. “Safe Reinforcement Learning on the Constraint Manifold: Theory and Applications”. In: *IEEE Transactions on Robotics* (2025). Early access.

[32] Matthew T. Mason. “Toward Robotic Manipulation”. In: *Annual Review of Control, Robotics, and Autonomous Systems* 1 (2018), pp. 1–28.

[33] Joseph Mirabel et al. “HPP: A New Software for Constrained Motion Planning”. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016.

[34] Joseph Mirabel et al. “Handling Implicit and Explicit Constraints in Manipulation Planning”. In: *Robotics: Science and Systems (RSS)*. 2018.

[35] Volodymyr Mnih et al. “Human-level Control Through Deep Reinforcement Learning”. In: *Nature* 518.7540 (2015), pp. 529–533.

[36] Tianqi Ni, Ahmed H. Qureshi, et al. *Physics-Informed Neural Motion Planning on Constraint Manifolds*. 2024.

[37] Johan Sebastian Obando-Ceron et al. "Revisiting the Lottery Ticket Hypothesis in Deep Reinforcement Learning through the Lens of Continual Learning". In: *arXiv preprint arXiv:2402.12479* (2024). arXiv: 2402.12479 [cs.LG].

[38] Nathan Ratliff et al. "CHOMP: Gradient Optimization Techniques for Efficient Motion Planning". In: *IEEE International Conference on Robotics and Automation (ICRA)* (2009), pp. 489–494.

[39] Isabel M. Rayas Fernández et al. *Learning Manifolds for Sequential Motion Planning*. 2020. arXiv: 2006.07746 [cs.RO].

[40] Amirreza Razmjoo et al. *Sampling-Based Constrained Motion Planning with Products of Experts*. 2024. arXiv: 2412.17462 [cs.RO].

[41] Nikita Rudin et al. "Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning". In: *Robotics: Science and Systems (RSS)*. 2022.

[42] John Schulman et al. "Trust Region Policy Optimization". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML)*. 2015, pp. 1889–1897.

[43] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *arXiv preprint* arXiv:1707.06347 (2017).

[44] Bruno Siciliano and Oussama Khatib. *Springer Handbook of Robotics*. Springer, 2008.

[45] Ghada A. Z. N. Sokar et al. "Dynamic Sparse Training for Deep Reinforcement Learning". In: *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI)*. 2022, pp. 3437–3443.

[46] Mike Stilman. "Manipulation Planning in High Dimensional Spaces Using Randomized Search". In: *ProceedINGS of the IEEE International Conference on Robotics and Automation (ICRA)*. Rome, Italy, 2007, pp. 345–350.

[47] Mike Stilman. "Global Manipulation Planning in Robot Joint Space with Task Constraints". In: *IEEE Transactions on Robotics* (2010).

[48] Ioan A. Şucan and Sachin Chitta. "Motion Planning with Constraints Using Configuration Space Approximations". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2012.

[49] Giovanni Sutanto et al. *Learning Equality Constraints for Motion Planning on Manifolds*. 2020. arXiv: 2009.11852 [cs.RO]. URL: https://arxiv.org/abs/2009.11852.

[50] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2nd ed. MIT Press, 2018.

[51] Chen Tessler, Yonathan Efroni, and Shie Mannor. "Action robust reinforcement learning and applications in continuous control". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 6215–6224.

[52] Marc Toussaint. *Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning*. Berlin, Germany: Springer, 2017.

[53] Marc A. Vischer, Robert T. Lange, and Henning Sprekeler. "On Lottery Tickets and Minimal Task Representations in Deep Reinforcement Learning". In: *arXiv preprint arXiv:2105.01648* (2021). arXiv: 2105.01648 [cs.LG].

[54] Marc Aurel Vischer, Robert Tjarko Lange, and Henning Sprekeler. *On Lottery Tickets and Minimal Task Representations in Deep Reinforcement Learning*. 2022. arXiv: 2105.01648 [cs.LG]. URL: https://arxiv.org/abs/2105.01648.

[55] Wendong Xiao et al. "Multimodal Fusion for Autonomous Navigation via Deep Reinforcement Learning with Sparse Rewards and Hindsight Experience Replay". In: *Displays* 81 (2023), p. 102523.

[56] Haonan Yu et al. "Playing the lottery with rewards and multiple languages: lottery tickets in RL and NLP". In: *ArXiv* abs/1906.02768 (2019). URL: https://api.semanticscholar.org/CorpusID:174801567.

[57] Kai Zhao et al. "Research on game-playing agents based on deep reinforcement learning". In: *Robotics* 11.2 (2022), p. 35.

# APPENDICES

# Appendix A

# Switching RL Algorithm to SAC

This chapter briefly discusses the proposed switch to the SAC algorithm, which is not implemented to maintain the LTH formation and the goal of creating a truly resource-efficient trainable agent. The discussion and results presented here, therefore, serve as a comparative reference rather than a proper methodological study and contribution.

## A.1 Brief about SAC

SAC is an off-policy actor-critic reinforcement learning algorithm designed for continuous action spaces. Unlike DDPG, which learns a deterministic policy, SAC employs a *stochastic* policy and optimizes a maximum-entropy objective. This formulation encourages policies that achieve high expected return while simultaneously maintaining high entropy, leading to improved exploration and robustness. Formally, SAC augments the standard reinforcement learning objective by incorporating an entropy regularization term:

$$J(\pi) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t \big(r(s_t, a_t) + \alpha\, \mathcal{H}(\pi(\cdot \mid s_t))\big)\right], \tag{A.1}$$

where $\mathcal{H}(\pi(\cdot \mid s))$ denotes the entropy of the policy at state $s$, and $\alpha > 0$ is a temperature parameter that controls the trade-off between reward maximization and exploration.

In contrast to deterministic policies of the form $a = \pi_\theta^D(s)$, the SAC actor learns a stochastic policy:

$$a \sim \pi_\theta(\cdot \mid s), \tag{A.2}$$

typically parameterized as a squashed Gaussian distribution. While this stochasticity enhances exploration, it also introduces fundamental differences in policy structure, gradient estimation, and network behavior when compared to DDPG.

## A.2 SAC Actor–Critic Formulation

As with DDPG, SAC follows an actor–critic architecture, but with key distinctions in both policy representation and value estimation.

**Soft Action-Value Function** The soft Q-function under policy $\pi$ is defined as:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^{t}\big(r(s_t,a_t) + \alpha\,\mathcal{H}(\pi(\cdot \mid s_t))\big) \,\middle|\, S_0 = s,\ A_0 = a\right]. \tag{A.3}$$

**Critic Learning** To reduce overestimation bias, SAC maintains two independent critic networks $Q_{\phi_1}$ and $Q_{\phi_2}$. The target for critic updates is computed as:

$$y = r + \gamma\Big(\min_{i\in\{1,2\}} Q_{\phi_i'}(s_k,a_k) - \alpha \log \pi_{\theta}(a_k \mid s_k)\Big), \tag{A.4}$$

where $a_k \sim \pi_{\theta}(\cdot \mid s_k)$ and $\phi_i'$ denote target network parameters.

**Actor Learning** The policy parameters $\theta$ are optimized by minimizing the Kullback-Leibler divergence [24][1] divergence between the policy and the exponential of the soft Q-function, resulting in the objective:

$$L_{\text{actor}}(\theta) = \mathbb{E}_{s\sim\mathcal{D},\,a\sim\pi_{\theta}}[\alpha \log \pi_{\theta}(a \mid s) - Q_{\phi}(s,a)]\,. \tag{A.5}$$

This stochastic optimization stands in direct contrast to the deterministic policy gradient used in DDPG, where gradients propagate exclusively through a single action selected by the actor network.

[1]The Kullback–Leibler (KL) divergence is a measure of how one probability distribution diverges from a reference distribution. Introduced by Kullback and Leibler in their seminal work [24].

## A.3 Challenges of Pruning Stochastic Policies

Despite its strong empirical performance, SAC is not integrated into the *Trikaya* framework due to its incompatibility with the core assumptions of the LTH. The stochastic nature of the policy introduces variability in action sampling that complicates weight-magnitude-based pruning and mask transfer across training runs.

In particular, the decision to prioritize DDPG over SAC in this work is driven by the need for computational efficiency within limited hardware resources. While SAC often provides superior exploration, its stochastic nature poses significant challenges when conserving processing power through sparsity. The existence of a stable, deterministic subnetwork, which is a central requirement for identifying robust lottery tickets, becomes difficult to guarantee when policy outputs are sampled from a learned distribution [53]. This stochasticity introduces variance in agent trajectories and gradient updates, destabilizing the convergence of weight importance measures. As a result, pruning masks obtained under SAC exhibit reduced reproducibility and diminished structural consistency when compared to those derived from deterministic DDPG policies [56], making the latter a more preferred candidate for resource-limited deployment, as shown in the visual Fig. A.1.

For these reasons, DDPG remains the primary algorithm used throughout this work, while SAC is explored only as a comparative baseline.

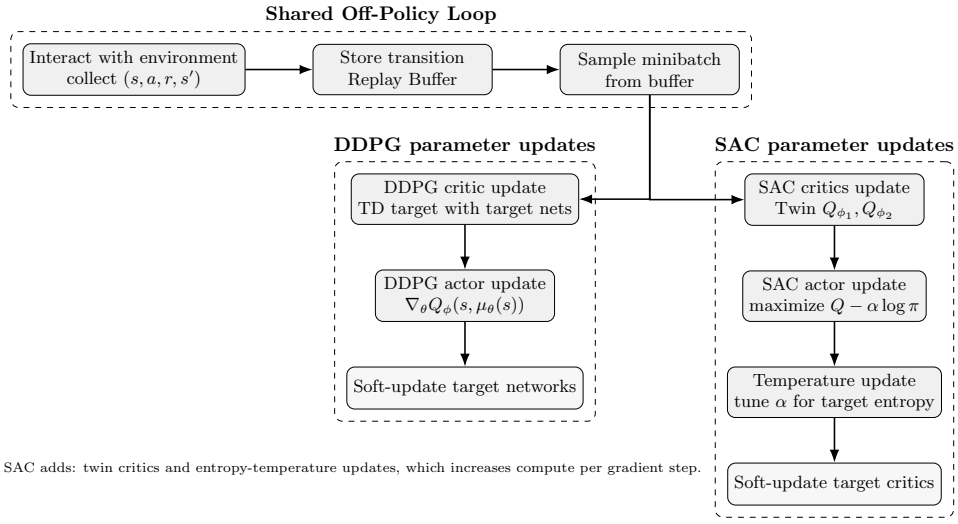



Figure A.1: Original training architecture of DDPG in a vertical layout.

## A.4 Results Achieved Using SAC

For completeness, SAC was evaluated across the same set of environments used in the main experimental Chapter 5. The results are summarized collectively to provide a high-level comparison of learning stability and performance.

### A.4.1 Circular Track Environment

The results in Fig. A.2 illustrate that the *Trikaya* framework remains effective when applied to SAC. The ticket network trained under Trikaya closely follows the performance of the full SAC agent, reaching similar return values with stable learning behaviour throughout training. This confirms that the framework generalizes naturally to stochastic and entropy-regularized policies, and that its pruning and constraint-aligned structure is not limited to deterministic learners.

Although SAC shows smoother learning curves and slightly improved exploration during early episodes, these benefits require significantly more computation. SAC relies on multiple critic networks, stochastic policy evaluations, and continuous entropy updates, which increase the training cost and reduce its compatibility with aggressive structured pruning. The overall performance remains comparable to the deterministic DDPG counterpart when Trikaya is applied. This outcome supports our decision to use DDPG as the primary algorithm in the *Trikaya* framework because it achieves similar results while offering faster training and far better resource efficiency.

### A.4.2 Planar Environment

The planar environment results follow the same success as the circular track setting as seen in Fig. A.3. Even when using SAC, the Trikaya ticket network converges reliably and maintains stable returns while operating under high sparsity. The appendix shows only one planar environment because it captures all the relevant behaviour needed to evaluate the framework. The key observation is that Trikaya continues to support efficient sparse training for SAC models, yet the additional computational requirements of SAC do not translate into a meaningful performance advantage.

### A.4.3 IIWA Hit Environment

The findings in Fig. A.4 extend this conclusion to a more complex robotic task. In the IIWA Hit environment, the Trikaya ticket agent again tracks the dense SAC model closely. The network

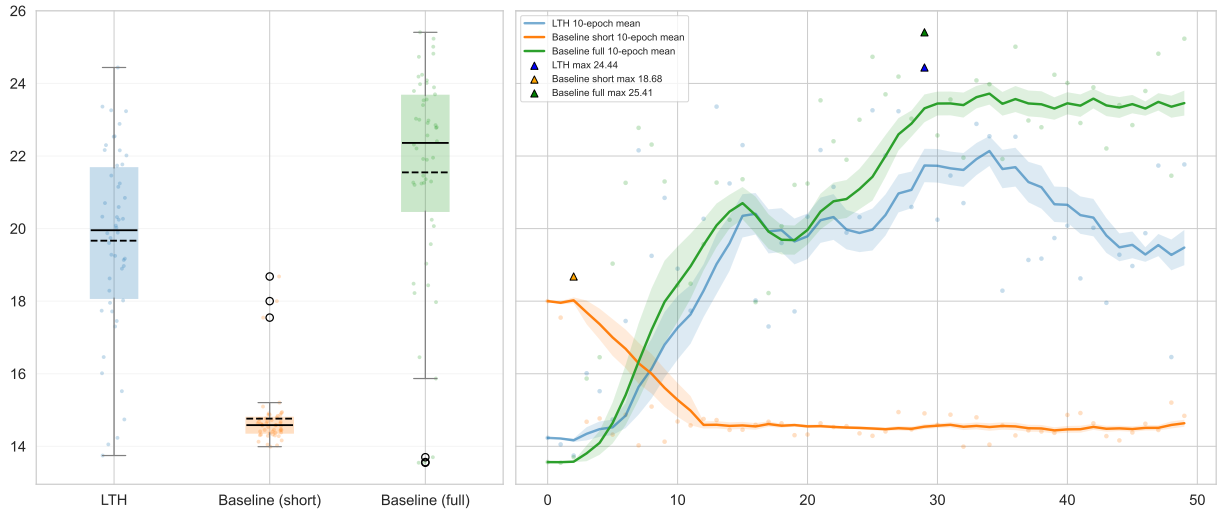



Figure A.2: Training performance for SAC based Trikaya on circular Environment comparing the ticket agent trained with the *Trikaya* framework (LTH; Blue) against two baselines (Orange: random short; Green: full). The $x$-axis shows training epochs, and the $y$-axis shows the relative discounted return $\mathbf{J}$. Left: reward distribution box plots. Right: 10-epoch moving-average learning curves with shaded empirical standard deviation.

maintains strong performance at high sparsity, and the learning curves remain smooth and consistent. This shows that Trikaya continues to scale effectively even when working with SAC in higher dimensional spaces.

The main limitation of SAC becomes more evident in this setting. Its higher computational load significantly increases training time, and yet the final performance of the ticket network remains comparable to that of a pruned DDPG agent. This confirms that the additional architectural components of SAC do not provide a practical benefit within the Trikaya framework. The deterministic approach remains preferable because it delivers similar performance while lowering the overall resource burden of training sparse networks.

### A.4.4 Summary

This demonstrates that the *Trikaya* framework remains fully applicable when paired with modern off-policy algorithms, and is capable of outperforming both dense and pruned baselines in achieving the overarching objective of efficient and stable learning. In particular, the framework continues to promote rapid convergence, reduced parameter counts, and robustness to sparsity, regardless of whether the underlying learner is deterministic or stochastic.

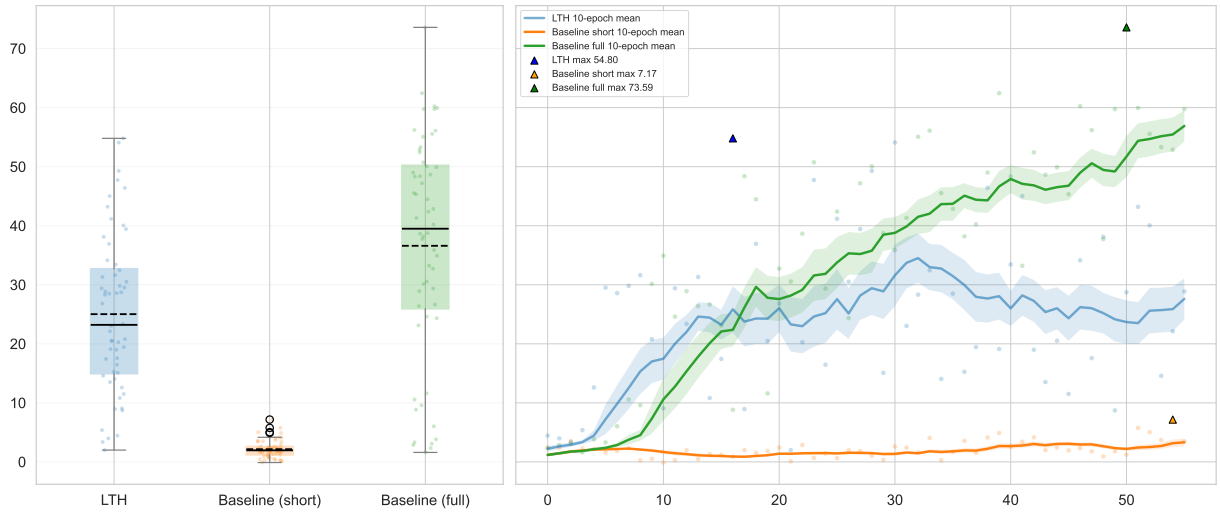



Figure A.3: Training performance for SAC based Trikaya on Planar Environment comparing the ticket agent trained with the *Trikaya* framework (LTH; Blue) against two baselines (Orange: random short; Green: full). The $x$-axis shows training epochs, and the $y$-axis shows the relative discounted return $\mathbf{J}$. Left: reward distribution box plots. Right: 10-epoch moving-average learning curves with shaded empirical standard deviation.

While SAC consistently exhibits smoother learning curves and improved exploration, more so during early training, these advantages come with notable trade-offs. The entropy-regularized objective requires maintaining multiple critic networks and evaluating stochastic policies, which increases computational overhead and makes the algorithm less compatible with aggressive structured pruning. As sparsity levels rise, SACs additional architectural components introduce sensitivity that is not present in deterministic approaches.

Taken together, these findings reinforce the design choice to prioritize deterministic actor–critic methods within the *Trikaya* framework. Deterministic learners benefit more naturally from constraint geometry, scale more cleanly under pruning, and maintain favourable computational properties, all while achieving comparable or superior performance across the evaluated tasks.

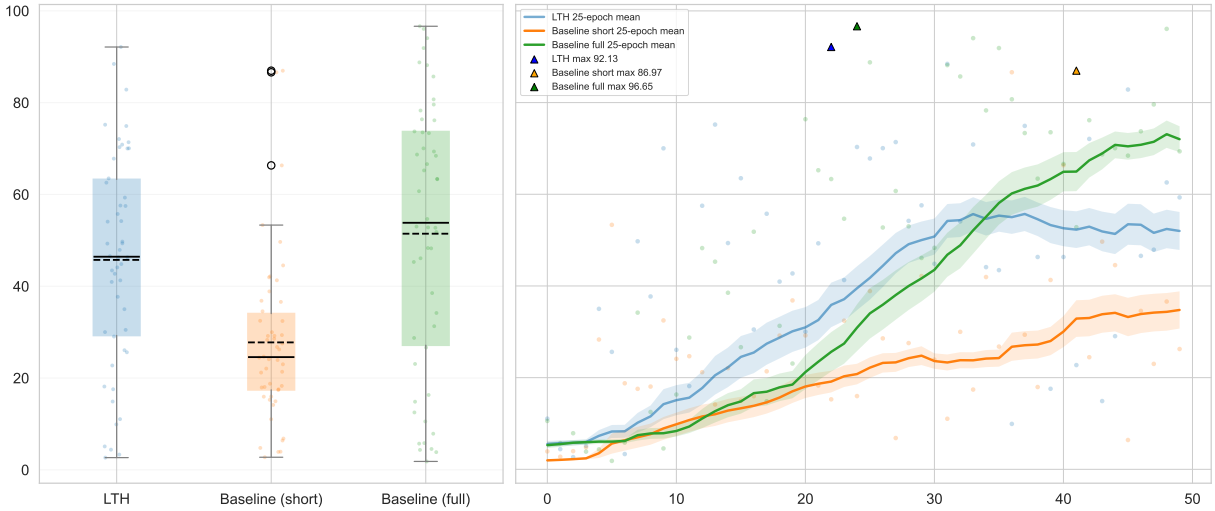



Figure A.4: Training performance for SAC based Trikaya on the IIWA hit environment comparing the ticket agent trained with the *Trikaya* framework (LTH; Blue) against two baselines (Orange: random short; Green: full). The $x$-axis shows training epochs, and the $y$-axis shows the relative discounted return $\mathbf{J}$. Left: reward distribution box plots. Right: 10-epoch moving-average learning curves with shaded empirical standard deviation.