MSc
Solving Long-Horizon Tasks via Imitation and Reinforcement Learning

by

Athanasia Lappa

Submitted
in partial fulfilment of the requirements for the degree of

Master of Artificial Intelligence

at the

UNIVERSITY OF PIRAEUS

February 2024

Author: Athanasia Lappa

II-MSc “Artificial Intelligence”

February 29, 2024

Certified by: George Vouros

George Vouros,
Professor,
University of Piraeus
Thesis Supervisor

Certified by: Maria Dagioglou

Maria Dagioglou,
Researcher,
NCSR Demokritos
Member of Examination Committee

Certified by: Orestis Telelis

Orestis Telelis,
Professor,
University of Piraeus
Member of Examination Committee

Solving Long-Horizon Tasks via Imitation and Reinforcement Learning

By

Athanasia Lappa

Submitted to the II-MSc "Artificial Intelligence" on February 29, 2024, in partial fulfillment of the requirements for the MSc degree

Abstract

This thesis explores the use of the Relay Policy Learning (RPL) algorithm proposed by Gupta et al. [1], to model trajectory prediction in an aviation environment. RPL is a two-phase approach consisting of a Hierarchical Imitation Learning (HIL) and Hierarchical Reinforcement Learning (HRL) algorithms. The purpose of this thesis is to model a policy learnt through RPL, to predict the aircraft trajectory. This is done through learning goal-conditioned hierarchical policies from unstructured and unsegmented demonstrations. This thesis utilizes a dataset with long aircraft trajectories. These are pre-processed to correct imperfections and to create low-level and high-level datasets from these demonstrations through the relay-data-relabelling augmentation of the RPL algorithm. Then the created datasets are used to learn hierarchical Imitation Learning (IL) policies without explicit goal labelling using the goal-conditioned Behavior Cloning (BC) method. This provides a policy initialization for subsequent relay reinforcement fine-tuning using a variant of the Trust Region Policy Optimization (TRPO) on-policy algorithm proposed by Schulman et al. [4]. Then, the implemented agent is tested and evaluated. The thesis concludes with a presentation of results and proposals for further work towards extending the RPL algorithm to work with off-policy RL algorithms.

Thesis Supervisor: George Vouros
Title: Professor, University of Piraeus

Εκτέλεση διαδικασιών μεγάλου χρονικού ορίζοντα με ενισχυτική μάθηση και μάθηση μέσω μίμησης

Από

Αθανασία Λάππα

Υποβλήθηκε στο ΔΠΜΣ «Τεχνητή Νοημοσύνη» την 29 Φεβρουαρίου 2024
ως υποχρέωση για την λήψη Μεταπτυχιακού Διπλώματος Σπουδών

Περίληψη

Αυτή η διπλωματική διερευνά τη χρήση του αλγορίθμου Relay Policy Learning (RPL) που προτείνεται από τους Gupta et al. [1], με στόχο την μοντελοποίηση της πρόβλεψης τροχιών αεροσκαφών, σε ένα αεροπορικό περιβάλλον. Ο αλγόριθμος RPL είναι μια προσέγγιση δύο φάσεων, στην πρώτη φάση χρησιμοποιεί έναν αλγόριθμο μάθησης με ιεραρχική μίμηση (Hierarchical Imitation Learning - HIL), ενώ στην δεύτερη φάση χρησιμοποιεί έναν αλγόριθμο ιεραρχικής ενισχυτικής μάθησης (Hierarchical Reinforcement Learning - HRL). Σκοπός αυτής της μεταπτυχιακής διπλωματικής εργασίας είναι να χρησιμοποιήσει τον εκπαιδευμένο πράκτορα από το RPL αλγόριθμο, για να προβλέψει την τροχιά ενός αεροσκάφους. Αρχικά, η εκπαίδευση του πράκτορα γίνεται με μη δομημένα δεδομένα, δηλαδή χωρίς να απαιτείται οι στόχοι του πράκτορα να έχουν καθοριστεί εκ των προτέρων. Η διατριβή χρησιμοποιεί ένα σύνολο δεδομένων με τροχιές αεροσκαφών. Αυτά υποβάλλονται σε προ-επεξεργασία για τη διόρθωση ατελειών και στην συνέχεια για τη δημιουργία συνόλων δεδομένων χαμηλού και υψηλού επιπέδου μέσω του αλγορίθμου επαύξησης δεδομένων (relay-data-relabelling augmentation) του RPL. Στην συνέχεια, τα σύνολα χαμηλού και υψηλού επιπέδου χρησιμοποιούνται για την εκμάθηση πολιτικών με μάθηση ιεραρχικής μίμησης (Hierarchical Imitation Learning - HIL), χρησιμοποιώντας έναν αλγόριθμο μίμησης βασισμένο σε στόχο (goal-conditioned Behavior Cloning – goal BC). Αυτό παρέχει μια αρχικοποίηση πολιτικής του πράκτορα για την επακόλουθη λεπτομερή εκμάθηση με χρήση του αλγόριθμου Trust Region Policy Optimization (TRPO) των Schulman et al. [4]. Στη συνέχεια, ο εκπαιδευμένος πράκτορας δοκιμάζεται και αξιολογείται. Η διπλωματική εργασία ολοκληρώνεται με μια παρουσίαση των αποτελεσμάτων και προτάσεις για περαιτέρω εργασία για την επέκταση του αλγορίθμου RPL με αλγόριθμους ενισχυτικής μάθησης εκτός πολιτικής (off-policy Reinforcement Learning).

Acknowledgments

First and foremost, I would like to thank my supervisor Professor George Vouros, for his continuous support, guidance, and advice throughout all this period that I have been working on my master's thesis. I should also thank him for providing me with access to the hardware resources of University of Piraeus.

I would also like to thank members of the AI-Lab (Christos Spatheris and Theocharis Kravaris) for helping me deal with subtle technical issues.

Finally, I want to express my deepest gratitude to my family for their continuous support throughout the years, as this thesis would not have been completed without their love and encouragement.

This thesis is dedicated in loving memory of my dearest mother,

Dimitra Lappa

Table of Contents

List of Figures

List of Tables

ABBREVIATIONS

ML = Machine Learning

DL = Deep Learning

ANN = Artificial Neural Network

DNN = Deep Neural Network

IM = Imitation Learning

HIL = Hierarchical Imitation Learning

RL = Reinforcement Learning

HRL = Hierarchical Reinforcement Learning

BC = Behavior Cloning

goal BC = goal Behavior Cloning

RPL = Relay Policy Learning

TRPO = Trust Region Policy Optimization

MLP = Multiple Layer Perceptron

MRM - Multilinear Regression Model

MSE – Mean Square Error

RELU – Rectified Linear Unit

1 Introduction

In this section, we motivate this thesis by mentioning the importance of utilizing Artificial Intelligence and Machine Learning (AI/ML) methods to determine aircraft's trajectories.

1.1 Machine Learning Overview

Nowadays, Artificial Intelligence (AI) is used everywhere to power intelligent applications. In brief, AI is the ability of a machine to somehow imitate, and maybe go beyond in some specific terms, intelligent human behavior. Machine Learning (ML) is a subset of the AI domain that allows the systems to make decisions without the need to be "manually" programmed in advance. ML methods can learn from data and understand the underlying patterns that are contained in them. Figure 1 illustrates the broad categories of ML methods.

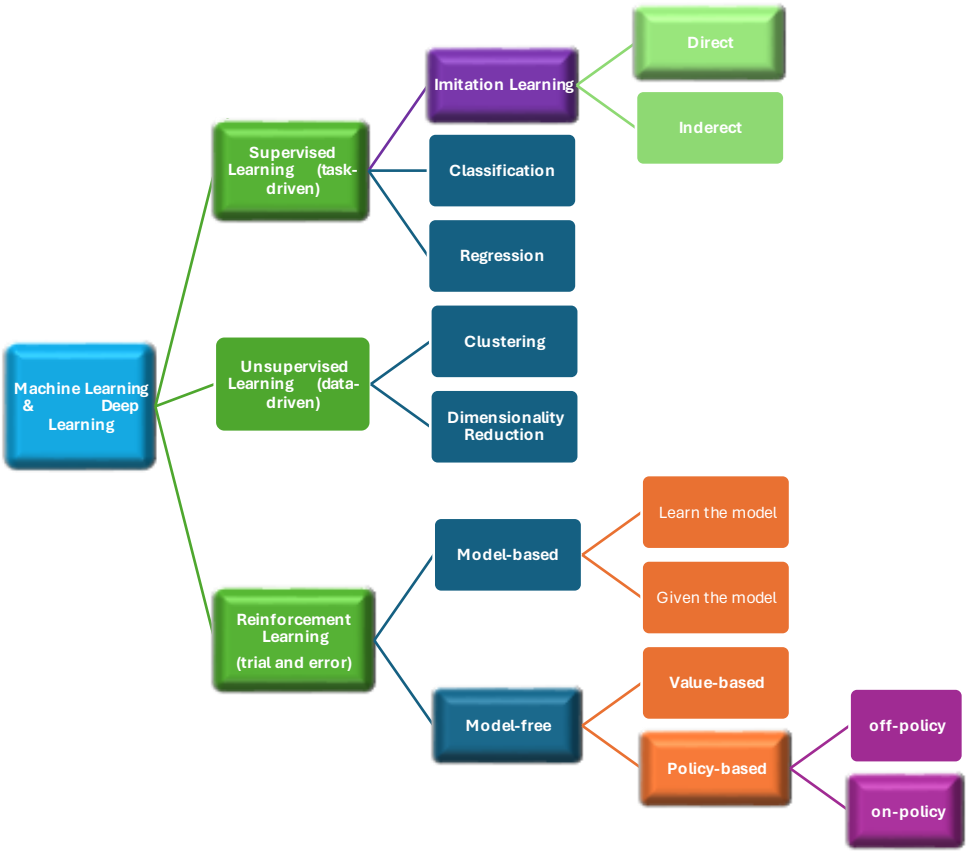

Figure 1: ML and DL main categories.

Deep Learning (DL) is a subset of ML. The main difference between the two is that in contrast to ML methods, where we apply one process that consists of a simple or more complex algorithm, in DL methods, multiple processes work together, forming layers, to capture the underlying representation of the data [35]. DL employs complex Deep Neural Networks (DNNs) that surpass the capabilities of machine learning methods. DNNs are essentially "function approximators". Deep Learning systems have been achieving incredible results in solving tasks in fields like Natural Language Processing (NLP), Computer Vision (CV), Robotics, real time decisions and many more.

ML and DL are broad fields but usually are classified into three main categories, Supervised Learning (SL), Unsupervised Learning (UL), and Reinforcement Learning (RL). DNNs can be used in supervised, unsupervised, or reinforcement learning, in various ways.

1.1.1 Supervised Learning

Supervised learning, also known as task-driven method, refers to models that can learn from labelled data to predict a value. It is the most common subbranch of ML and is usually classified into two main categories, Classification and Regression algorithms.

Classification

The **classification** approach refers to the modelling problem of predicting a discrete class label output for a given input instance. Some widely used classification machine learning algorithms are Support Vector Machines (SVMs), Logistic Regression, Decision Tree, Naïve Bayes Classifier, and K-Nearest Neighbors (KNNs). Algorithms of the classification category are widely used for image classification, fraud detection, email spam detection and diagnostics.

Regression

The **regression** approach refers to the modelling problem when the target variable is a real or continuous value that needs to be approximated. Among the most popular regression machine learning algorithms are Linear Regression (LR), Lasso Regression, Support Vector Regression (SVR) and Multivariate Regression. Algorithms of the regression category are widely utilized for risk assessment and score prediction.

Imitation Learning

Imitation Learning (IL) is a form of supervised learning, also known as learning from demonstrations, and refers to an intelligent agent who receives desired behavior demonstrations from an expert and attempts to perform a task conforming to the expert's behavior. IL may assume that the expert policy is optimal. The expert demonstrations may originate from humans or even from other agents that perform actions to complete a specific task. The agent can be any machine learning model. However, due to multi-dimensionality and continuity of the state-action space, it usually incorporates Deep Neural Networks (DNNs) which can efficiently approximate different complex models, e.g. a policy model.

IL is usually classified into two main categories, direct and indirect imitation.

Direct

The **direct** imitation approach refers to an agent who directly learns how to imitate the expert's policy. The Behavior Cloning (BC), the Direct Policy Learning (DPL), and the Dataset Aggregation (DAgger) algorithm are classical approaches that focus on directly imitating the policy. However, DAgger is difficult to use in practice as it requires access to an expert during all the training, rather than just a set of demonstrations.

Indirect

The **indirect** imitation approach refers to an agent who indirectly imitates the policy by learning the expert's reward function, which is commonly referred to as Inverse Reinforcement Learning (IRL). In addition to other tasks, IRL has been employed in navigation [24], autonomous driving [25], and manipulation [26].

1.1.2 Unsupervised Learning

Unsupervised learning, also known as data-driven method, refers to models that can learn from unlabeled data, without predefined labels on the dataset, by finding patterns in data. It is usually classified into two main categories, Clustering and Dimensionality Reduction algorithms.

Clustering

The **clustering** approach attempts to find patterns in data and separates them into multiple subgroups based on the similarity. The most famous clustering algorithm is k-means algorithm. Other algorithms are the density-based, distribution-based and hierarchical-based. Algorithms of the clustering category are widely used for city planning, targeted marketing, and biology.

Dimensionality Reduction

The **dimensionality Reduction** approach attempts to extract low-dimensional features from the original data set. Some dimensionality reduction algorithms are Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA). Algorithms of the dimensionality reduction category are widely used for image recognition, face recognition, text mining, and big data visualization.

1.1.3 Reinforcement Learning

Reinforcement Learning (RL), also known as trial-and-error method, refers to intelligent agents who are trained to achieve goals through a recurring interaction process in a stochastic and potentially complex environment. Every time the agent chooses and performs a certain action that changes the state of the environment. The agent receives either rewards or penalties for the actions it performs. The agent's goal is to learn an optimal policy which maximizes the long-term cumulative rewards. Algorithms of the reinforcement learning category are widely and successfully used for AI gaming, robot navigation, real time decisions, learning tasks, and skill acquisition.

RL is usually classified into two main categories, model-based and model-free algorithms.

Model-based.

In the **model-based** approach, the agent can either learn or have access to a model of the environment. By a model of the environment, we mean a function which predicts state transitions and rewards [34]. Then the agent collects data to update the model. It can be a greedy algorithm with the aim to maximize the reward at each step. It is suitable in situations where we have complete knowledge about the environment, even in situations where no rewards are available.

However, the main downside is that most model-based algorithms, beyond being very inefficient during training, are over-fitting.

Model-based methods are usually classified into two main categories, “Learn the model” and “Given the model”.

Learn the model.

Algorithms that belong to “Learn the model” category are World Models, Imagination-Augmented Agent (I2A) [36], Model-Based Priors for Model-Free (MBMF) [37], Generative Pretrained Transformers – GPT 3.5 + (who learn the reward function to optimize performance).

Given the model.

An algorithm of the “Given the model” category is the AlphaZero [34].

Model-free.

In the **model-free** approach the agent directly updates a learned value function or policy through interaction with the environment. Model-free methods can successfully solve various tasks but require many samples to achieve good performance. This approach is suitable in environments with a dynamic nature and where we cannot have sufficient knowledge, even after many interactions. For example, autonomous driving cars have a dynamic environment where there can be numerous changes in traffic routes.

Model-free methods are usually classified into two main categories, value-based and policy-based algorithms.

Value-based algorithms

Algorithms belonging to this approach are Deep Q Learning (DQN), Quantile-Regression Deep Q Learning (QR-DQN), and Hindsight Experience Replay (HER) [38].

Policy-based algorithms

Policy-based methods, e.g. policy gradient methods, are usually classified into two main categories, off-policy, and on-policy algorithms.

Off-policy algorithms

Off-policy algorithms are using a different policy for acting and training. They can reuse previous data very efficiently. Some widely used off-policy algorithms are Deep Deterministic Policy Gradient (DDPG), Twin Delayed DDPG(TD3) and Soft Actor Critic (SAC).

On-policy algorithms

On-policy algorithms are using the same policy for acting and training. They don't use previous data, which makes them weaker on sample efficiency. Among the most popular on-policy algorithms are Vanilla Policy Gradient (VPG), Trusted Region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO).

1.2 Scope of thesis

The Machine Learning method's categories that are studied in this thesis are the direct imitation supervised learning method and the on-policy, model-free RL method, which are primarily studied on aircraft trajectory prediction.

1.2.1 Machine Learning in the Aviation Domain

In the aviation domain, predicting aircrafts' trajectories can be challenging. Due to it being a long-horizon task, it requires extended exploration. A key objective is to develop learning methods to implement robust, autonomous intelligent agents, that can perform well in complex, real-world aircraft navigation.

1.2.2 Motivation

Gupta et al. propose the Relay Policy Learning (RPL) algorithm [1], a method for imitation and reinforcement learning that can solve multi-stage, long-horizon robotic tasks. Gupta et al. in [1], devised a simple and universally applicable two-phase approach that in the first phase pre-trains hierarchical policies using demonstrations such that they can be easily fine-tuned using RL during the second phase. Authors demonstrate the effectiveness of their method on several multi-stages, long-horizon manipulation tasks in a challenging kitchen simulation environment. Aircraft trajectory prediction is a multi-stage, long-horizon task, assuming a sparse binary reward.

1.2.3 Contribution

The main topic of this thesis is aircraft trajectory prediction in the aviation domain, by implementing and evaluating a variant of the RPL algorithm. The aircraft trajectories employed are from Paris to Istanbul. Each trajectory indicates the movement of the aircraft, the most important observations are

coordinates of aircraft position at any time step, i.e., longitude, latitude, and altitude.

1.3 Structure of thesis

This thesis is subdivided into the following different topics:

Chapter 1: Introduction refers to a brief overview of the AI and the AI in aviation domain, highlighting the scope and objectives of the current thesis.

Chapter 2: Background and Related Work provides a brief introduction of the research areas of Imitation and Reinforcement Learning and a closer look at the combination of the above methods. Furthermore, provides a brief overview of Relay Policy Learning (RPL) algorithm proposed by Gupta et al [1].

Chapter 3: Methodology presents in detail the demonstrations dataset used in our problem setting. It presents the preprocessing steps, and it further discusses the proposed approach based on RPL algorithm [1],

Chapter 4: Experiments and Results contains the experimental setup and the performance measures used for the evaluation of the agent.

Chapter 5: Conclusion and Future work are a summary of the accomplished work as well as a proposal of future research opportunities.

Bibliography provides a list of sources referred to in this thesis, to further facilitate reader's access to the selected articles and books.

2 Background and Related Work

This section provides a brief introduction to Artificial Intelligence research areas of Imitation and Reinforcement Learning, and specifically to the Relay Policy Learning (RPL) algorithm proposed by Gupta et al. [1].

2.1 Imitation Learning (IL)

It has long been known that humans and animals use imitation as a mechanism for acquiring knowledge. In the context of Artificial Intelligence (AI), Imitation Learning (IL) refers to a family of supervised machine-learning methods, which can be used to quickly generate a rough solution to a given task, using demonstrated behavior.

However, IL is not a recent advancement in Machine Learning (ML). One of the first applications was carried out in 1989 by Dean Pomerleau. Specifically, Behavior Cloning (BC) algorithm was used to train ALVINN, one of the first self-driving cars [19]. Since then, IL techniques, where expert demonstrations of good behavior are used to train a policy, have proven very useful in practice, and have led to state-of-the-art performance in a variety of applications [3].

As mentioned earlier, the IL is a form of supervised learning. Firstly, the intelligent agent is provided with a set of input features, the independent variables. Then, it is trained to predict a target variable, the dependent variable. During the training process, when the expert provides demonstrations to the agent, the agent receives both the input features and the target variable. The input features are the agent's state observations, while the target variable is agent's action. Therefore, the training data is composed of state-action pairs. The agent observes the state of the environment, and the actions demonstrated by the expert, and frames a policy based on it.

2.1.1 Imitation Learning formalism

In IL formalism, the agent has access to demonstrations D containing a set of trajectories $D = \{\tau^i, \tau^j, \tau^k, \dots\}$ of state-action pairs $\tau^i = \{s_0^i, a_0^i, s_1^i, a_1^i, \dots s_T^i, a_T^i\}$.

Markov Decision Process (MDP) formalism for IL

It is assumed that the environment in IL is a Markov Decision Process (MDP) with states s and actions a, and the set of admissible states and actions are referred to as S and A. The reward function $r(s,a)$ in R is unknown. The system dynamics are expressed by the probabilistic transition model:

$$p(s_{t+1}|s_t, a_t)$$

The transition function which is the conditional probability distribution over s_{t+1}, given a state and action at a particular times step.

2.1.2 Imitation Learning challenges.

In classical Supervised Learning, each state-action pair is assumed to be independent of others and follows a specific distribution.

In Imitation Learning the agent's environment, which is modelled by a Markov Decision Process (MDP), given an action in each state, induces the next state, which breaks the crucial Independent and Identically Distributed (IID) assumption. IID is a fundamental assumption of almost all statistical learning approaches, meaning that each of the training data points used to build a model need to be independent of each other and are randomly sampled from the same underlying distribution.

2.1.3 Behavior Cloning (BC)

A straightforward and common approach to imitation learning is Behavior Cloning (BC), which focuses on learning the expert's policy using Supervised Learning, as said above. BC learns a policy through learning to mimic the demonstrated, state-action pairs. The objective is to learn an optimal policy π*(a|s) by imitating the demonstrations. Here, being optimal, the aim is to maximize the likelihood of actions demonstrated:

$$E_{(s,a)\sim D} log\pi(a|s)$$

In some tasks, BC can be utilized effectively. Its main advantages are its simplicity and efficiency. Suitable applications can be either short-horizon tasks where expert's trajectories can cover the observation space or tasks where committing an error doesn't result in catastrophic failures.

However, for most of the tasks, BC can be quite challenging and is often unable to perform well across all temporally extended tasks due to compound errors. The primary reason for this is that it violates the fundamental Independent and Identically Distributed (IID) assumption [3] as mentioned in section 2.1.2.

2.1.4 Hierarchical Imitation Learning (HIL)

Hierarchical Imitation Learning (HIL) is a potential solution to solve sparse reward issues in challenging long-horizon tasks, by introducing temporal abstraction. HIL try to achieve two goals, learn a temporal task abstraction, and discover a meaningful segmentation of the demonstrations into subtasks.

Goal-conditioned formalism for HIL

When there are multiple demonstrated tasks, we consider a goal-conditioned imitation learning setup where the dataset of demonstrations D contains sequences that attempt to reach different goals $s_g^i, s_g^j, s_g^k, \dots$

The objective is to learn a goal-conditioned optimal policy $\pi^*(a|s, s_g)$ that can reach different goals s_g by imitating the demonstrations.

2.2 Reinforcement Learning (RL)

Trial and error are an essential approach to solving problems in which multiple attempts are made to achieve a solution. In the context of Artificial Intelligence (AI), Reinforcement Learning (RL) refers to an intelligent agent who interacts with an environment and receives rewards and penalties. The agent's goal is to learn an optimal policy which maximizes the long-term cumulative rewards. In detail, we define the interaction rules, and the agent explores different paths and possibilities.

The RL interaction process includes receiving a reward and the next state of the environment every time when the agent chooses and performs a certain action at a specific state, enabling state transition. The agent's environment is either

fully or partially observable. Figure 2 illustrates the RL iterative process of agent-environment interaction to form trajectories and solve sequential problems.

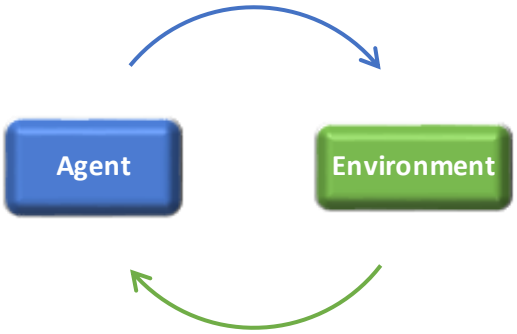

Figure 2: The iterative interaction process.

In general, the recurring RL interaction process makes it possible to incorporate uncertainty and typically, solving the following types of tasks:

- *Tasks of achievement*, such as "close the door", can be represented by giving a positive reward for achieving the goal.
- *Tasks of prevention*, such as "don't crash with another vehicle", can be represented by giving a negative reward when bad events occur.
- *Tasks of maintenance*, such as "keep the air-traffic control system working as long as possible", giving a positive reward for each time step that the desirable state is maintained [16].

2.2.1 Reinforcement Learning formalism.

The formulation of the Reinforcement Learning (RL) problem is similar to the Imitation Learning (IL) problem. The main difference is that in RL, instead of having access to demonstrations D, the agent gets rewards $r(s, a)$.

RL algorithms address the problem of how an intelligent agent can learn to approximate a strategy while interacting directly with its environment. The goal of RL is to find an optimal policy $\pi^*(a|s)$ that maximizes expected reward over trajectories induced by the policy:

$$E_\pi[\Sigma_{t=0}^{\mathrm{T}} \gamma^t r_i(s_t, a_t)]$$

The variable γ is the discounting factor, which controls to what degree rewards in the distant future affect the total value of a policy and is usually just slightly less than 1, $0 \leq \gamma < 1$.

Markov Decision Process (MDP) formalism for RL

Most Reinforcement Learning (RL) research is based on the formalism of problems as Markov Decision Processes (MDP). Although RL is by no means restricted to MDP, this discrete-time, finite state and action formalism provides the simplest framework in which to study algorithms.

A finite MDP models the following type of problem. We define $\mathrm{M} = (S, A, P, r, \rho_0, \gamma)$ to be a finite-horizon Markov decision process (MDP), where S and A are state and action spaces, $p(s_{t+1}|s_t a_t)$ is the transition probability of each state known as transition function, $r(s, a)$ is the reward function, ρ_0 is the distribution of the initial states of the trajectories and γ is the discount factor.

MDP follows the Markovian property that s_{t+1} depends only on previous state information.

2.2.2 Reinforcement Learning challenges.

Reinforcement Learning (RL) has achieved significant success in many cases but has been largely confined to relatively simple short-horizon tasks.

One of the challenges of Reinforcement Learning (RL) is the exploration-exploitation trade-off. The exploration refers to the agent's actions that may lead to new information and potentially higher rewards in the future. The exploitation refers to the agent's actions that have high expected rewards based on its current knowledge.

Another challenge is the curse of dimensionality, there are long-horizon tasks where the possible states and actions in a complex environment can be very large to effectively execute the costly iterative RL process. Furthermore, the environment may not be fully observable to be able to explore all the possible paths towards a goal state.

Finally, many long-horizon, multi-step tasks with continuous control are natural to specify with a sparse reward. To address this issue, one can manually design rewards functions, which provide the agent with more frequent rewards. Moreover, in some cases such as self-driving car, there isn't any direct reward

function, thus, the approach of manual shaping a reward function is necessary. However, manually designing a reward function that satisfies the desired behavior can be extremely complicated and can result in suboptimal performance.

These challenges of RL which come from the exploration-exploitation trade-off, the curse of curse of dimensionality and the sparse and complicated reward function, puts many real-world tasks out of practical reach of RL methods.

2.2.3 Deep Reinforcement Learning

Deep Reinforcement Learning (deep RL), a combination of RL and Deep Learning (DL), is a potential solution to solve the curse of dimensionality in challenging long-horizon tasks. Deep RL algorithms can employ Deep Neural Networks (DNNs) since they can efficiently approximate the policy or the reward function. This allows them to generalize across the observation space so that the learning time scales much better. The DNN is trained at every training iteration by updating its parameters. There can be DNN models either value-based or policy-based. Figure 3 illustrates an example of deep RL.

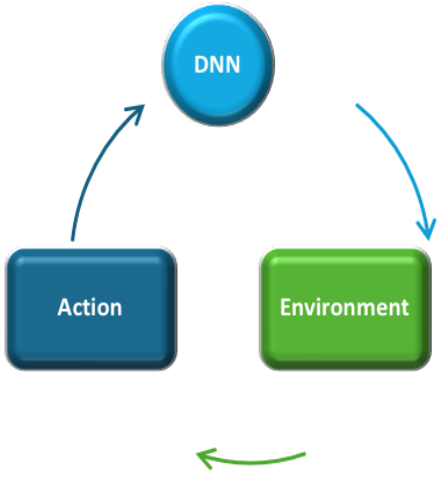

Figure 3: The deep RL model that trains a DNN to get action from state inputs.

Recent years, deep RL have accumulated significant and many times impressive results to numerous challenging domains such as Atari games [17] and Go [18].

2.2.4 Hierarchical Reinforcement Learning (HRL)

Hierarchical Reinforcement Learning (HRL) is a potential solution to solve sparse reward issues in challenging long-horizon tasks, by introducing temporal abstraction. Thus, decisions are not required at each step, but rather invoke the execution of temporally extended activities which follow specific policies until termination. This leads to hierarchical control architectures and associated learning algorithms [5].

Barto et al. [5] review several related approaches to temporal abstraction and hierarchical organization that machine learning researchers have developed: the "options formalism" of Sutton, Precup, and Singh [13], the Hierarchies of Abstract Machines (HAMs) approach of Parr and Russell [14,15], and Dietterich's MAXQ framework [16]. The MAXQ framework provides a hierarchical decomposition of the given RL problem into a set of subproblems [16]. Common to these approaches is a reliance on semi-Markov Decision Processes (semi-MDP) [5].

A review of relative research shows that there are several important design decisions that must be made when constructing an HRL method [16]. In detail, HRL involves breaking the target Markov decision problem into a hierarchy of subproblems or subtasks. There are three general approaches to defining these subtasks:

- One approach is to define each subtask in terms of a fixed policy that is provided by the programmer. The "options formalism" of Sutton, Precup, and Singh [13] takes this approach.
- The second approach is to define each subtask in terms of a non-deterministic finite-state controller. The Hierarchy of Abstract Machines (HAM) method of Parr and Russell [14, 15] takes this approach. This method permits the programmer to provide a "partial policy" that constrains the set of permitted actions at each point but does not specify a complete policy for each subtask.
- The third approach is to define each subtask in terms of a termination predicate and a local reward function. These define what it means for the subtask to be completed and what the final reward should be for

completing the subtask. The MAXQ method of Dietterich takes this approach [16].

Goal-conditioned formalism for HRL

To extend Reinforcement Learning (RL) to multiple tasks, a goal-conditioned formulation presented by Leslie Pack Kaelbling [9], can be used to learn a policy $\pi(a|s, s_g)$ which maximizes the expected reward $r(a, s, s_g)$ with respect to a goal distribution $s_g \sim G$ as follows:

$$\mathrm{E}_{s_g \sim G}[\mathrm{E}_{\pi}[\Sigma_{t=0}^{T} \gamma^t r_i(s_t, a_t, s_g)]]$$

2.3 Combination of Imitation and Reinforcement Learning

As mentioned in 2.1.2 section, the well-known compounding error stemming from Imitation Learning (IL) is often unable to perform well in long-horizon tasks [3]. Recent research has demonstrated that the Reinforcement Learning (RL) approach is a potential solution to this issue, by enabling continuous improvement of the learned policy from experience. The RL approach is employed to improve IL policies through fine-tuning.

However, the use of Imitation Learning (IL) to bootstrap the process of Reinforcement Learning (RL) has been previously utilized by several deep RL algorithms. The bootstrapping helps to overcome exploration challenges, while RL fine-tuning allows the policy to improve based on actual task objective.

Rajeswaran et al. [6] propose learning complex dexterous manipulation with Deep Reinforcement Learning (DRL) and demonstrations. They propose to augment the policy search process with a small number of human demonstrations collected in virtual reality (VR). They found that pre-training a policy with Behavior Cloning (BC), and subsequent fine-tuning with policy gradient along with an augmented loss to stay close to the demonstrations, dramatically reduces the sample complexity, enabling training within the equivalent of a few real-world robot hours.

Zhu et al. [7] propose a model-free deep reinforcement learning method that leverages a small amount of demonstration data to assist a reinforcement agent.

They apply this approach to robotic manipulation tasks and train end-to-end visuomotor policies that map directly from RGB camera inputs to joint velocities.

Nair et al. [8] propose overcoming exploration in environments with sparse rewards in RL with demonstrations. Their method, which builds on top of Deep Deterministic Policy Gradients and Hindsight Experience Replay [38], provides an order of magnitude of speedup over RL on simulated robotics tasks.

The above approaches demonstrate that agents combining imitation and reinforcement significantly improved performance than agents trained with RL or IL alone. However, these approaches include a flat IL initialization that is improved using reinforcement learning with additional auxiliary objectives [6,7,8]. The flat algorithms treat the state space as a huge flat search space. This means that the paths from the start state to the goal state are very long, and the length of these paths determines the cost of learning and planning, as information about future rewards must be propagated backward along these paths.

Gupta et al. [1] demonstrate the Relay Policy Learning (RPL) method, which is described in the next chapter, where agents can use to learn hierarchical policies in a way that can be fine-tuned better than their flat counterparts.

2.4 Relay Policy Learning (RPL)

As mentioned throughout the previous chapters, solving multi-stage, long-horizon robotic tasks can be challenging. To tackle these problems, Gupta et al. propose the Relay Policy Learning (RPL) algorithm, a simple and universally – applicable two-phase hierarchical approach, consisting of an imitation learning phase that produces goal-conditioned hierarchical policies, and a hierarchical reinforcement learning phase that finetunes these policies for task performance [1].

In contrast to Hierarchical Reinforcement Learning (HRL) methods, the RPL method takes advantage of unstructured demonstrations to bootstrap further fine-tuning, and in contrast to conventional Hierarchical Imitation Learning (HIL) methods, it does not focus on careful subtask segmentation, but instead splits the demonstration data into fixed-length segments. This simplification allows them to leverage the idea of relabelling demonstrations across different goals. The RPL authors [1] demonstrate the effectiveness of their method on

several multi-stage, long-horizon manipulation tasks in a challenging kitchen simulation environment.

According to the RPL authors [1] the main advantage of their approach is that it is simple and very general, in that it can be applied to any demonstrated data, including easy to provide unsegmented, and unstructured demonstrations of meaningful behaviours. Furthermore, this method does not require any explicit form of skill segmentation or subgoal definition, which otherwise would need to be learned or explicitly provided. Lastly, and most importantly, since this method ensures that every low-level trajectory is goal-conditioned and of the same limited length, it is very amenable to reinforcement fine-tuning, which allows for continuous policy improvement.

2.4.1 Relay Policy Learning architecture.

The algorithm starts with unstructured, unlabelled demonstrations D, which correspond to meaningful activities provided by the user. The pool of demonstrations consists of N trajectories $D = \{\tau_0, \tau_1, \tau_2, \dots \tau_N\}$, where each trajectory consists of state-action pairs $\tau = \{s_1, a_1, s_2, a_2, \dots s_T, a_T\}$. Importantly, these demonstrations can be attempting to reach a variety of different high-level subgoals, but do not require these subgoals to be specified explicitly.

To take the most advantage of such data, the authors pre-trains goal-conditioned hierarchical policies using the proposed **Relay Imitation Leaning (RIL)** algorithm, which construct low-level and high-level datasets from the demonstrations, and then use them to perform imitation learning. This provides a good policy initialization for subsequent **Relay Reinforcement Fine-tuning (RRF)**. Figure 4 illustrates the Relay Policy Learning algorithm [1].

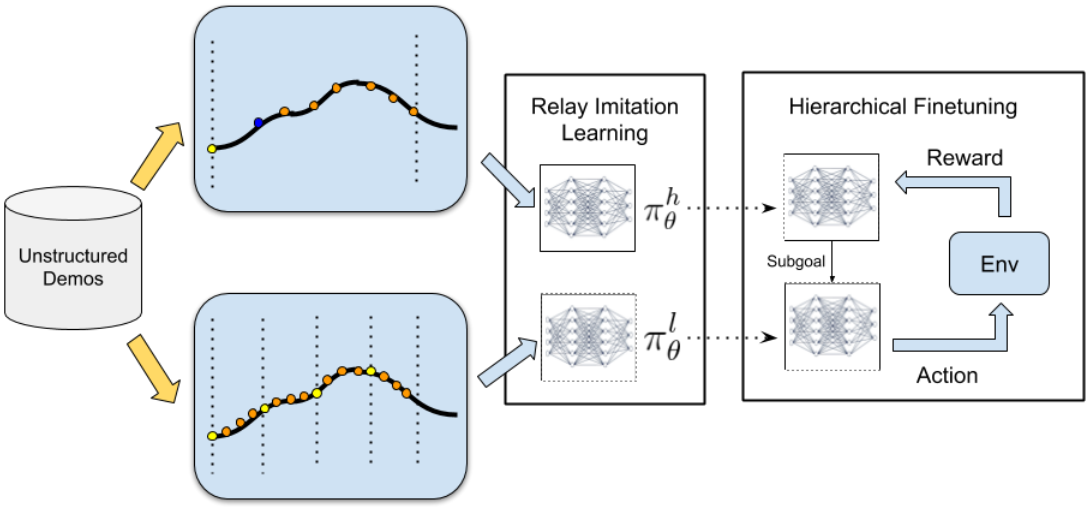

Figure 4: Relay Policy Learning algorithm [1]

Figure 5 illustrates the RPL [1] hierarchical policy architecture, which is composed of a high-level policy and a low-level policy, which together generate an action at a given state. High level Policy sets subgoals for low-level policy [2]. Low-level policy takes that subgoal and output low level actions to act in the environment [2]. Only low-level act to the environment.

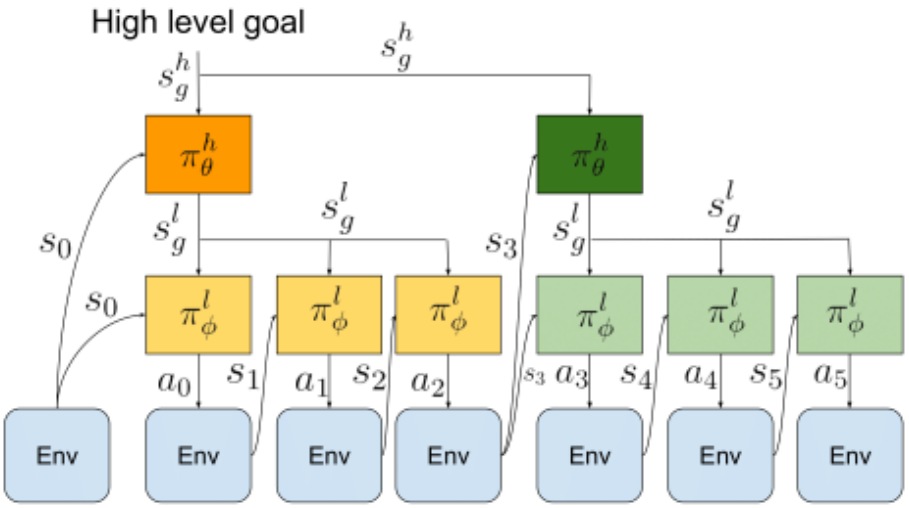

Figure 5: Relay Policy Learning architecture [1].

Initially, the high-level policy takes the current state and creates a high-level subgoal that is passed to the low-level policy. Then, the low-level policy takes the current state and the subgoal created by the high-level policy to create an action which is executed in the environment. For the subsequent H time steps, set to 30 in [1], the subgoal created by the high-level policy is kept fixed, while the low-level policy takes the current state and creates an action at every time step.

The overall steps of the Relay Policy Learning (RPL) method [1] are described in Algorithm 1 in figure 6.

Algorithm 1 Relay Policy Learning

Require: Unstructured pool of demonstrations $D = \{\tau_0, \tau_1, ...\tau_N\}$

1: Relabel goals in demonstration trajectories using Algorithm 2, 3 to extract D_l, D_h
2: **Relay Imitation Learning:** Train π_θ^h and π_ϕ^l using Eqn 1
3: **while** not done **do**
4: Collect on-policy experience with π_θ^h and π_ϕ^l for high level goals different s_g^h
5: [Optional] Relabel this experience (Sec. 4.3), and add to D_l, D_h
6: Update the policy via policy gradient update using Eqn 2, 3.
7: **end while**
8: Distill fine-tuned policies into a single multi-goal policy

Figure 6: Overview of steps for RPL algorithm [1].

2.4.2 Relay Imitation Learning (RIL)

To learn the relay policy from meaningful but unstructured demonstrations D, the RPL authors devise the relay data relabelling augmentation algorithm to construct a low-level dataset D_l and a high-level dataset D_h from the demonstrations, and then use these datasets to perform imitation learning.

Relay data relabelling augmentation algorithm.

Gupta et al. [1] present a novel relay data-relabelling augmentation algorithm for learning goal-conditioned hierarchical policies. In detail, they construct a low-level dataset D_l and a high-level dataset D_h by iterating through the pool of demonstrations D and use them to learn the high-level hierarchical policy π_θ^h and the low-level hierarchical policy π_φ^l via supervised learning at multiple levels.

According to the RPL authors [1], the relay data relabelling augmentation algorithm does not only enable us to learn hierarchical policies without explicit labels, but also provides algorithmic improvements to imitation learning. Firstly, generates more data through the relay data relabelling augmentation algorithm. Secondly, it improves generalization since it is trained on a large variety of subgoals.

Construct the low-level dataset D_l

RPL [1] iterates through the pool of demonstrations D to construct the low-level dataset D_l. Firstly, the method is configured with a low-level window size W_l and it generates state-action-goal (s, a, s_g^l) tuples for low-level dataset D_l within a sliding window size W_l along the demonstrations, as described in Algorithm 2 in figure 7.

Algorithm 2 Relay data relabeling for RIL low level

Require: Demonstrations $D = \{\tau_0, \tau_1, \ldots \tau_N\}$

for $n = 1 \ldots N$ **do**
  for $t = 1 \ldots t_n$ **do**
    for $w = 1 \ldots W_l$ **do**
      Add $(s_t^n, a_t^n, s_{t+w}^n)$ to D_l
    end for
  end for
end for

Figure 7: Overview of steps to construct the low-level dataset [1].

The key idea behind this is to consider all states that are reached along a demonstration trajectory within W_l time steps from any state S_t to be goals reachable from the state S_t by executing action a_t, without the requirement for any explicit goal labelling from a human demonstrator [1].

For example, consider the following trajectory τ_0 from the pool of demonstration $D = \{\tau_0, \tau_1, \tau_2, \ldots \tau_N\}$ consisting of N trajectories:

$$\tau_0 = \{s_1, a_1, s_2, a_2, s_3, a_3, s_4, a_4, s_5, a_5, s_6, a_6, s_7, a_7, s_8, a_8, s_9, a_9, s_{10}, a_{10}, \ldots s_T a_T\}$$

For each state-action (s, a) pair in τ_0 state-action-goal (s, a, s_g^l) tuples are created. If we set the low-level window size to be six, $W_l = 6$, the created labels for the (s_1, a_1) pair are the following:

$$s_1, a_1, s_2$$

$$s_1, a_1, s_3$$

$$s_1, a_1, s_4$$

$$s_1, a_1, s_5$$

$$s_1, a_1, s_6$$

Repeating this procedure for all state-action pairs in the τ_0 trajectory.

The authors in [1] tried to utilize different low-level window sizes for RPL. Their ablations suggest that the larger the window, the harder the learning problem becomes for both, imitation, and RL fine-tuning. Finally, they chose the low-level window size W_l to be 30-time steps in all their experiments.

Construct the high-level dataset.

RPL [1] employ a similar procedure to construct the high-level dataset D_h. Firstly, they choose a high-level window size W_h and then generate state-action(subgoal)-goal tuples for high-level dataset D_h, within the sliding window size W_h along the demonstrations, as described in Algorithm 3 in figure 8.

Algorithm 3 Relay data relabeling for RIL high level

Require: Demonstrations $D = \{\tau_0, \tau_1, \ldots \tau_N\}$

for $n = 1 \ldots N$ **do**
  for $t = 1 \ldots t_n$ **do**
    for $w = 1 \ldots W_h$ **do**
      Add $(s_t^n, s_{t+\min(w, W_l)}^n, s_{t+w}^n)$ to D_h
    end for
  end for
end for

Figure 8: Overview of steps to construct the high-level dataset [1].

The high-level action (subgoal state) is set to j steps ahead s_{t+j}, as $s_{t+\min(W_l, j)}$ choosing a sufficiently distant subgoal as the high-level action (subgoal state).

For example, consider the following trajectory τ_0 from the pool of demonstration $D = \{\tau_0, \tau_1, \tau_2, \ldots \tau_N\}$ consisting of N trajectories:

$$\tau_0 = \{s_1, a_1, s_2, a_2, s_3, a_3, s_4, a_4, s_5, a_5, s_6, a_6, s_7, a_7, s_8, a_8, s_9, a_9, s_{10}, a_{10}, \ldots s_T a_T\}$$

For each state-action (s, a) pair in τ_0 trajectory creates state-action(subgoal)-goal tuples. If we set the high-level window size to be nine, $W_h = 9$, and set the high-level action (subgoal state) to be j steps ahead s_{t+j}, as $s_{t+\min(W_l, j)}$, the j will be six as the low-level window size. So, for state s_1, high-level window size $W_h = 9$, and j=6 the subgoal will be the s_7. In detail, the created labels will be the following:

$$s_1, s_7, s_2$$

$$s_1, s_7, s_3$$

$$s_1, s_7, s_4$$

$$s_1, s_7, s_5$$

$$s_1, s_7, s_6$$

$$s_1, s_7, s_7$$

$$s_1, s_7, s_8$$

Repeating this procedure for all state-action pairs in τ_0 trajectory.

In [1], high-level window size W_h is set to 260 in all the experiments.

Imitation Learning

According to the RPL authors [1], the Relay Imitation Learning (RIL) algorithm is a simple imitation learning procedure that builds on the goal relabelling scheme described in Lynch et al. [33] for the hierarchical setting, resulting in improved handling of multi-task generalization and compounding error.

Given these relay-data-relabelled datasets, they train a high-level policy π_θ^h and a low-level policy π_φ^l by maximizing the likelihood of the actions taken given the corresponding states and goals.

$$max_{\varphi,\theta} \ \mathrm{E}_{(s,a,s_g^l)\sim D_l}\left[log\pi_\varphi\left(a\middle|s,s_g^l\right)\right] + \mathrm{E}_{(s,s_g^l,s_g^h)\sim D_h}[log\pi_\vartheta(s_g^l|s,s_g^h)]$$

Equation 1: Relay Imitation Learning equation [1].

In practice, this is a goal-conditioned Behaviour Cloning (BC) for the low-level and the high-level datasets.

The RPL authors [1], used in their experiments Multilayer Perceptron (MLP) feed-forward artificial neural networks, with two layers of 256 units each and ReLu nonlinearities for both the high-level policy π_ϱ^h and the low-level policy π_φ^l. All imitation learning algorithms use the ADAM optimizer using a batch size of 128 and a learning rate of 0.005.

2.4.3 Relay Reinforcement Fine-tuning (RRF)

Gupta et al. [1] employ a goal-conditioned Hierarchical Reinforcement Learning (HRL) algorithm for fine-tuning the extracted policies from the RIL phase. The algorithm used is a variant of Trust Region Policy Optimization (TRPO) proposed by Schulman and al. [4]. In detail, a variant of Natural Policy

Gradient (NPG) with adaptive step, where both the high-level and the low-level goal-conditioned policies π_θ^h and π_φ^l are being trained with policy gradient in a decoupled optimization [1].

In detail, given a low-level goal-reaching reward function $r_l(s_t, a_t, s_g^l)$, we can optimize the low-level policy by simply augmenting the state of the agent with the goal commanded by the high-level policy and then optimizing the policy to effectively reach the commanded goals by maximizing the sum of its rewards [1]. For the high-level policy, given a high-level goal-reaching reward function $r_h(s_t, g_t, s_g^h)$, authors in [1] optimize it by running a similar goal-conditioned policy gradient optimization to maximize the sum of high-level rewards obtained by commanding the current low-level policy.

To encourage extracted policies at both levels from Relay Imitation Learning (RIL) phase to stay close to the behaviour shown in the demonstrations, the Natural Policy Gradient (NPG) objective is augmented with a max-likelihood objective that ensures that policies at both levels take actions that are consistent with the relabelled demonstration pools D_l and D_h from relay data relabelling algorithm, as described in Equation 2 and 3 [1]:

$$\nabla_\varphi J_l = \mathrm{E}\left[\nabla_\varphi log\pi_\varphi^i(a|s, s_g^l)\sum_t r_l(s_t, a_t, s_g^l)\right] + \lambda_l E_{(s,a,s_g^l)\sim D_l}\left[\nabla_\varphi log\pi_\varphi^l(a|s, s_g^l)\right]$$

Equation 2: Reinforcement learning (low-level).

$$\nabla_\vartheta J_h = \mathrm{E}\left[\nabla_\theta log\pi_\theta^h(s_g^l|s, s_g^h)\sum_t r_h(s_t, s_g^l, s_g^h)\right] + \lambda_h E_{(s,s_g^l,s_g^h)\sim D_h}\left[\nabla_\vartheta log\pi_\theta^h(s_g^l|s, s_g^h)\right]$$

Equation 3: Reinforcement learning (high-level).

The RPL authors [1] fine-tune on 17 different compound goals individually, with a path length of 260 for every compound goal, and the low-level horizon set to 30. They use 100 trajectories in each iteration of on-policy fine-tuning, with a discount of 0.995. When using variants of augmenting the policy gradient objective with demonstrations, they experimented with different weights λ_h and λ_l, but they found 0.0001 to work well. They use a batch size of a 100 trajectories per iteration, and standard parameters for truncated natural policy gradient

based on python package [11], which contains implementations of various RL algorithms for continuous control tasks simulated with MuJoCo.

3 Methodology

In this chapter, we describe our attempt to reengineer the Relay Policy Learning (RPL) algorithm [1] and evaluate its use to predict aircraft trajectories.

3.1 Demonstrations Datasets

In this section, the datasets that were utilized are being presented along with the proposed method of data-preparation.

The datasets contain airplane traffic data collected from Paris to Istanbul route, in Comma Separated Value (CSV) files. The CSV files are referred to as "the dataset", and they are the starting point of the Relay Imitation Learning (RIL). The CSV file holds information about the trajectories of aircraft. Each row in the file corresponds to twenty (20) seconds, and the state space is naturally discretized into twenty-seconds time steps.

The dataset contains the longitude, latitude, and the altitude of each time step. Some names of the parameters in the dataset are not self-explanatory. Therefore, table (1) explains the relevant parameters and airplane terminologies related to them.

Table 1: Dataset

	Parameter name	Parameter Description	Datatype
1	trajectory_ID	Airplane trajectory ID, a combination of date and ID.	object
2	longitude	Longitude of the specific timestamp.	float64
3	latitude	Latitude of the specific timestamp.	float64
4	altitude	Altitude of the specific timestamp.	Float64
5	timestamp	Time stamp in Unix type.	int64
6	temp_iso	Temperature	Float64
7	v_wind_component	Vertical wind direction, a positive value means that the direction of the airflow is upward, while a negative	Float64

		value denotes that the direction is downward.	
8	u_wind_component	Horizontal wind direction.	Float64
9	Cluster	Airplane trajectory route.	int64
10	model_id	Airplane model ID.	int64
11	dlon	Longitude difference of the current timestamp with the longitude of the next timestamp.	Float64
12	dlat	Latitude difference of the current timestamp with the latitude of the next timestamp.	Float64
13	dalt	Altitude difference of the current altitude with the altitude of the next timestamp.	Float64
14	delay	Airplane delay.	int64

3.1.1 Overview training dataset.

The training dataset contains 64,245 rows from 116 trajectories, which are distributed in six separate routes. The diagram in figure 9 illustrates the separate routes. As we can notice the shortest route – dataset column "Cluster" - is this with value five (5).

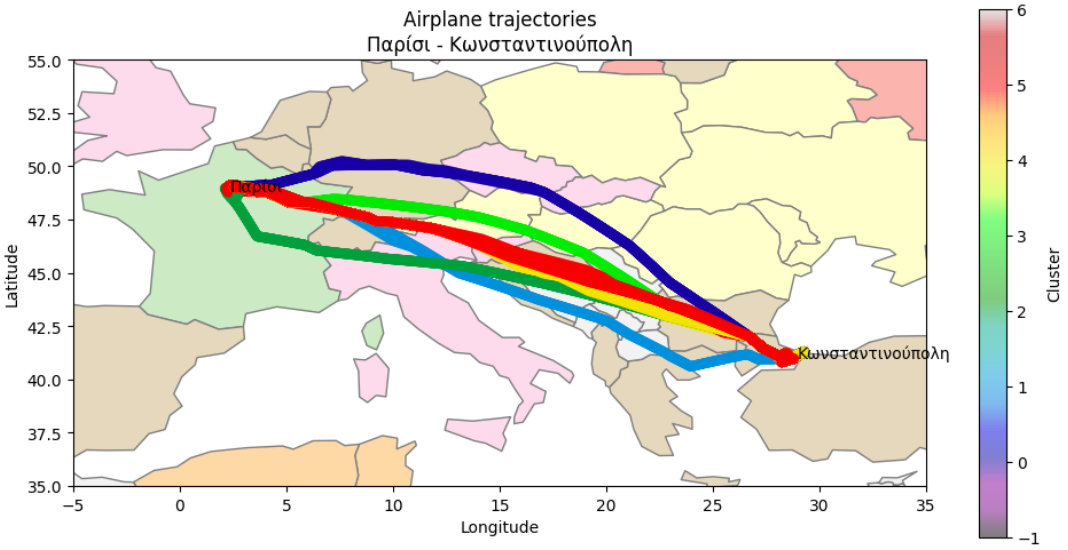

Figure 9: Airplane training trajectories.

The diagram in figure 10 illustrates the distribution of trajectories across the different routes. The route of cluster five (5) represents 72% of the whole dataset, so this was selected to train the agent.

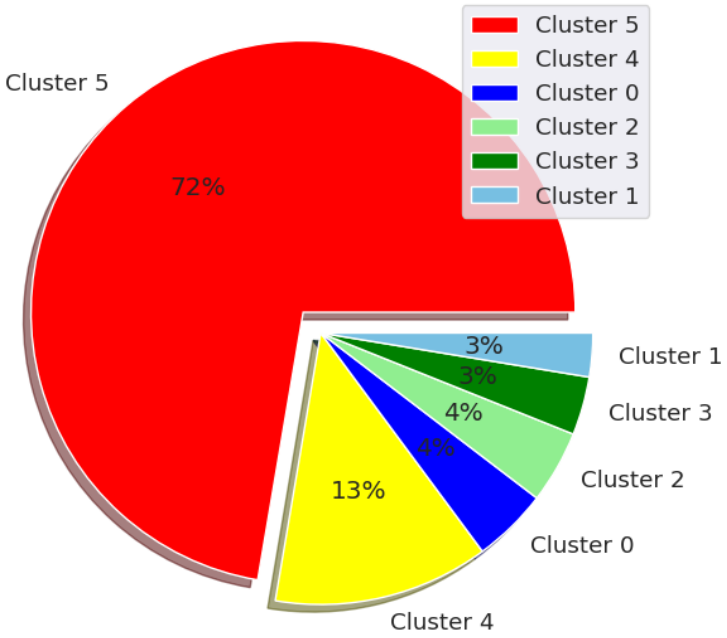

Figure 10: Distribution of training trajectories.

3.1.2 Cleaning training dataset

Convert trajectory ID to numerical value.

The trajectory ID in the dataset is a combination of date and an ID. Therefore, we cropped the date and kept only the ID.

Remove trajectory routes.

Removed from the dataset the other trajectories and kept only trajectories in cluster with value 5 in agent's training.

Convert Unix timestamp.

Convert Unix timestamp to date – time format with seconds.

Check for missing data.

Each row in the train dataset corresponds to a twenty (20) seconds time-interval update, as we can notice in the data variable timestamp. So, for each trajectory ID we computed the time duration in minutes. The diagram in figure 11 illustrates the total time steps and the time duration in minutes for each trajectory ID.

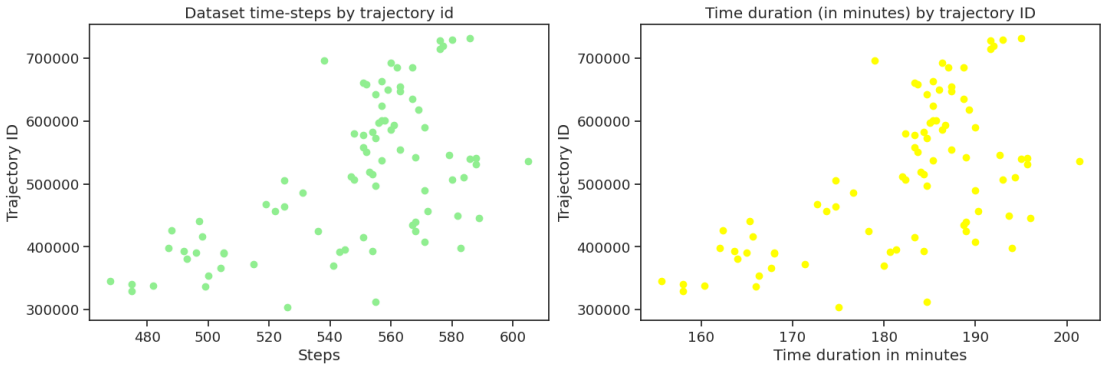

Figure 11: Check for missing data.

Overview training dataset after the data cleaning process.

After the data cleaning process, the training dataset contains 46,467 rows from 85 trajectories in route - data field “Cluster” - five (5). The diagram in figure 12 illustrates the altitude differences between the trajectories in route five (5).

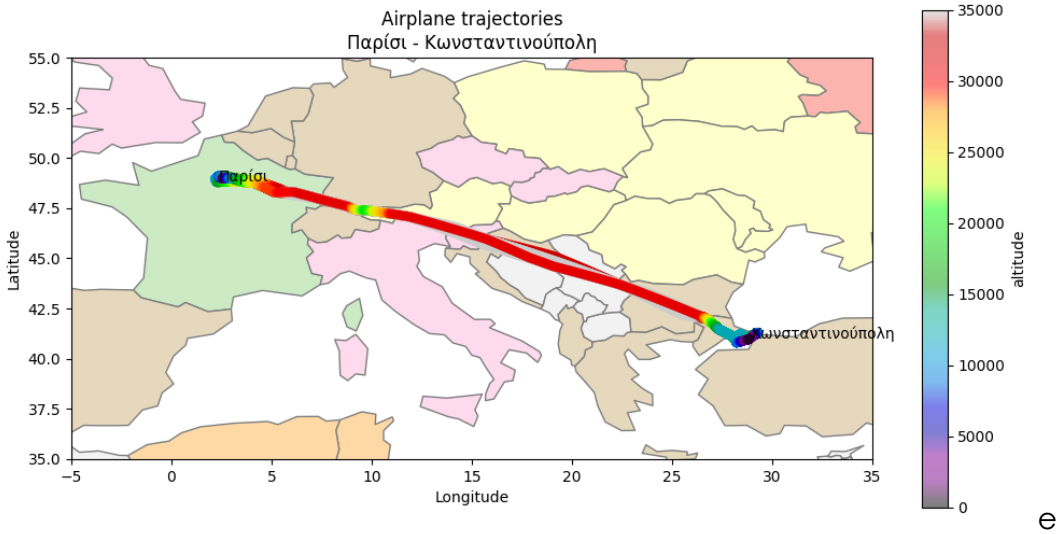

Figure 12: Training trajectories route 5 altitude.

Figure 13 illustrates trajectory ID cardinality in training dataset.

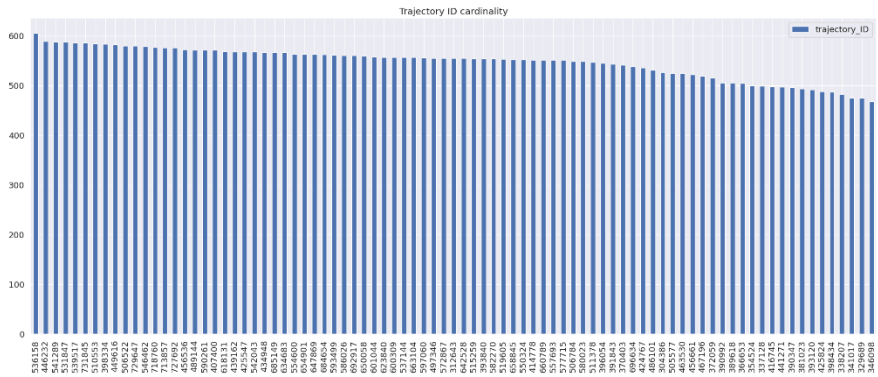

Figure 13: Trajectory ID cardinality in training dataset.

3.1.3 The Training Dataset

Data Correlation

Data correlation refers to the statistical relationship between two data variables. We computed the Pearson correlation coefficient [39], which is the most common measurement for a linear relationship between two variables. The stronger the correlation between these two variables, the closer it will be to +1 or -1. A correlation coefficient of -1 describes inverse correlation, with values in one series rising as those in the other decline, and vice versa. The diagram in figure 14 illustrates correlation between variables of the dataset.

Figure 14: Pearson correlation coefficient.

Obviously, the longitude and the latitude variables clearly have a negative correlation (-0.99), this indicates that the two variables move in opposite directions. The latitude of Paris, France is 48.864716, and the longitude is 2.349014, while the latitude of Istanbul, Turkey is 41.015137, and the longitude is 28.979530.

Negative correlation exists between temperature and altitude variables (-0.72), when the altitude increases the temperature decrease and vice versa. Also, there is negative correlation between the longitude selection – dataset column "dlon" - and the temperature (-0.63).

The longitude selection – data column "dlon" – and the altitude have a positive correlation (0.79), this indicates the two variables move either up or down in the same direction together. Also, there exists a positive correlation between altitude selection – data column "dalt" - and the latitude (0.61).

All the other dataset variables have zero or small correlation, which indicates that there is no relationship between them and that they are considered being unrelated.

Longitude selection correlation

The diagrams in figure 15 illustrate the correlations between the longitude and the longitude, latitude, altitude, and temperature variables.

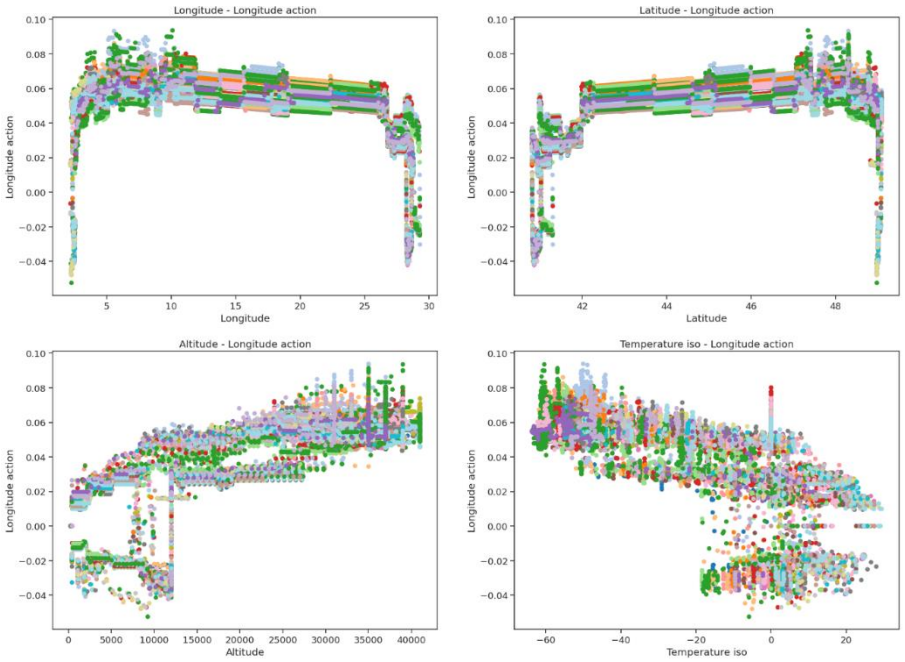

Figure 15: Longitude selection correlation.

Latitude selection correlation

The diagram in figure 16 illustrates the correlations between the latitude and the longitude, latitude, altitude, and temperature variables.

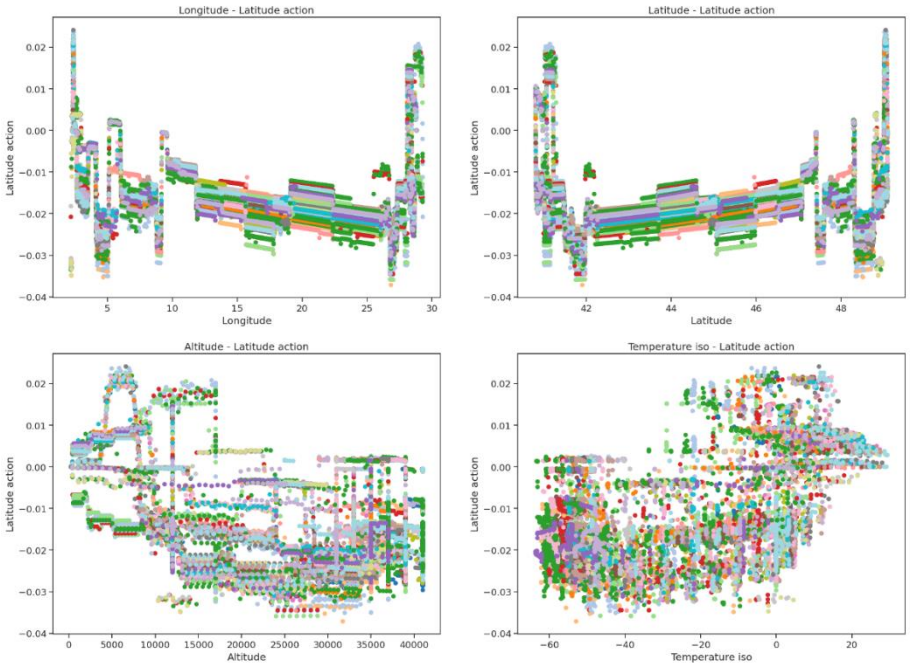

Figure 16: Latitude selection correlation.

Altitude selection correlation

The diagram in figure 17 illustrates the correlations between the altitude and the longitude, latitude, altitude, and temperature variables.

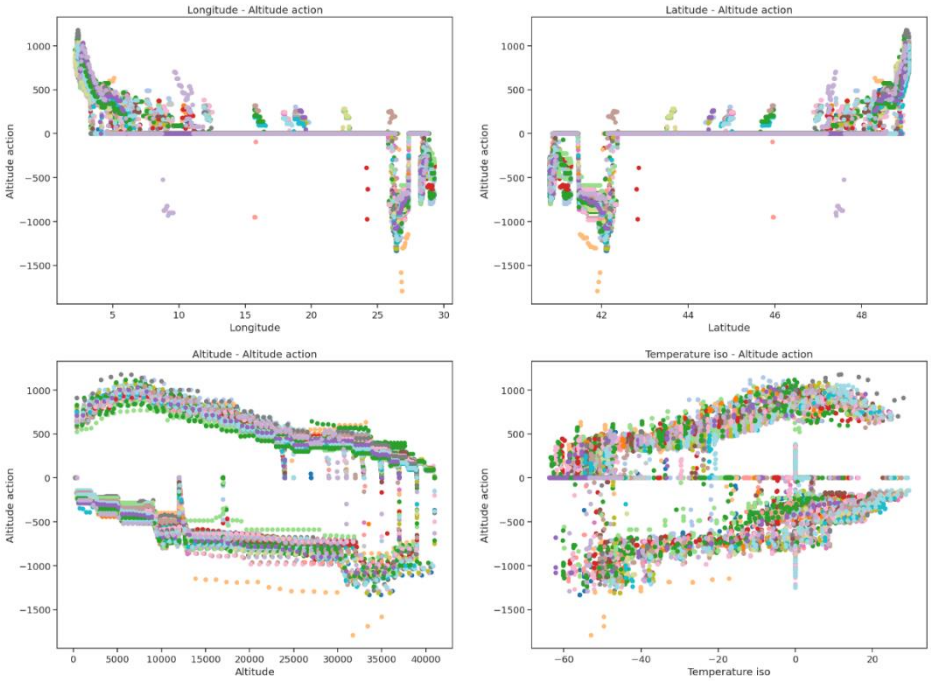

Figure 17: Altitude selection correlation.

Longitude, Latitude and Altitude distribution

The diagram in figure 18 illustrates the distribution of the longitude, latitude, and altitude dataset variables.

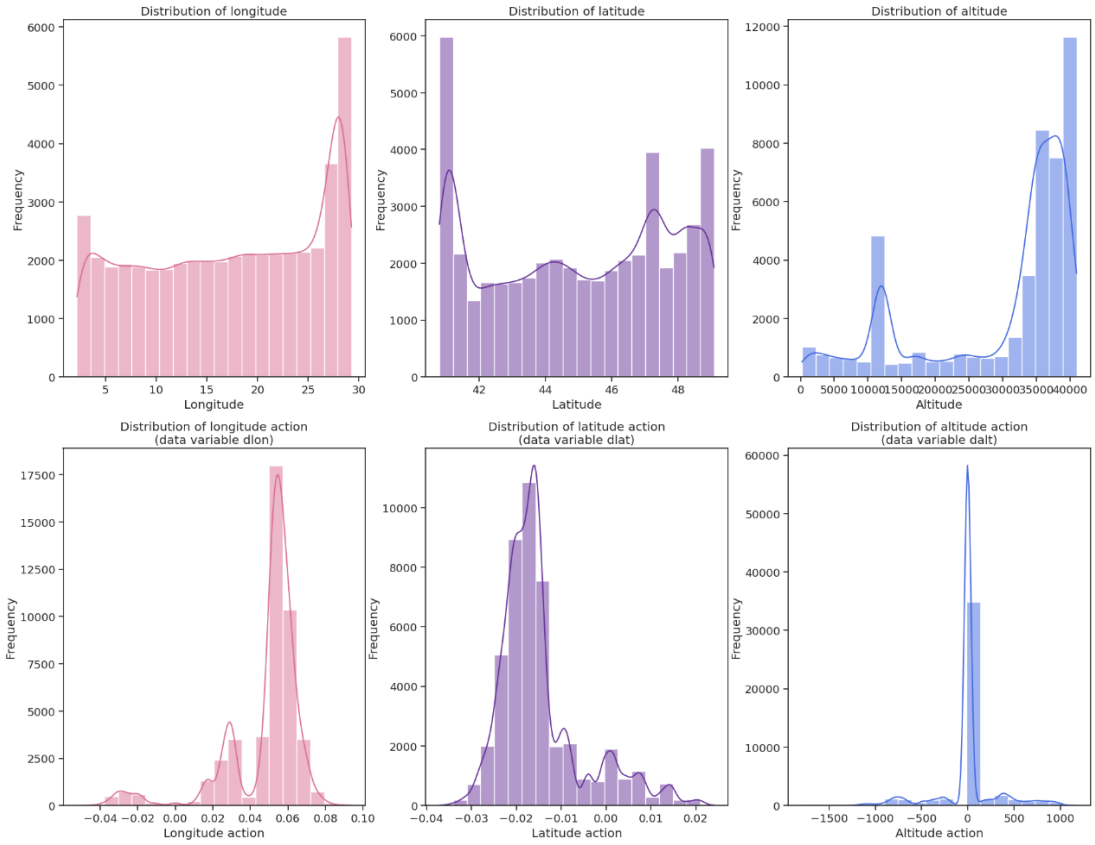

Figure 18: Longitude, Latitude and Altitude distribution.

Training trajectories

The diagram in figure 19 illustrates the 85 training trajectories.

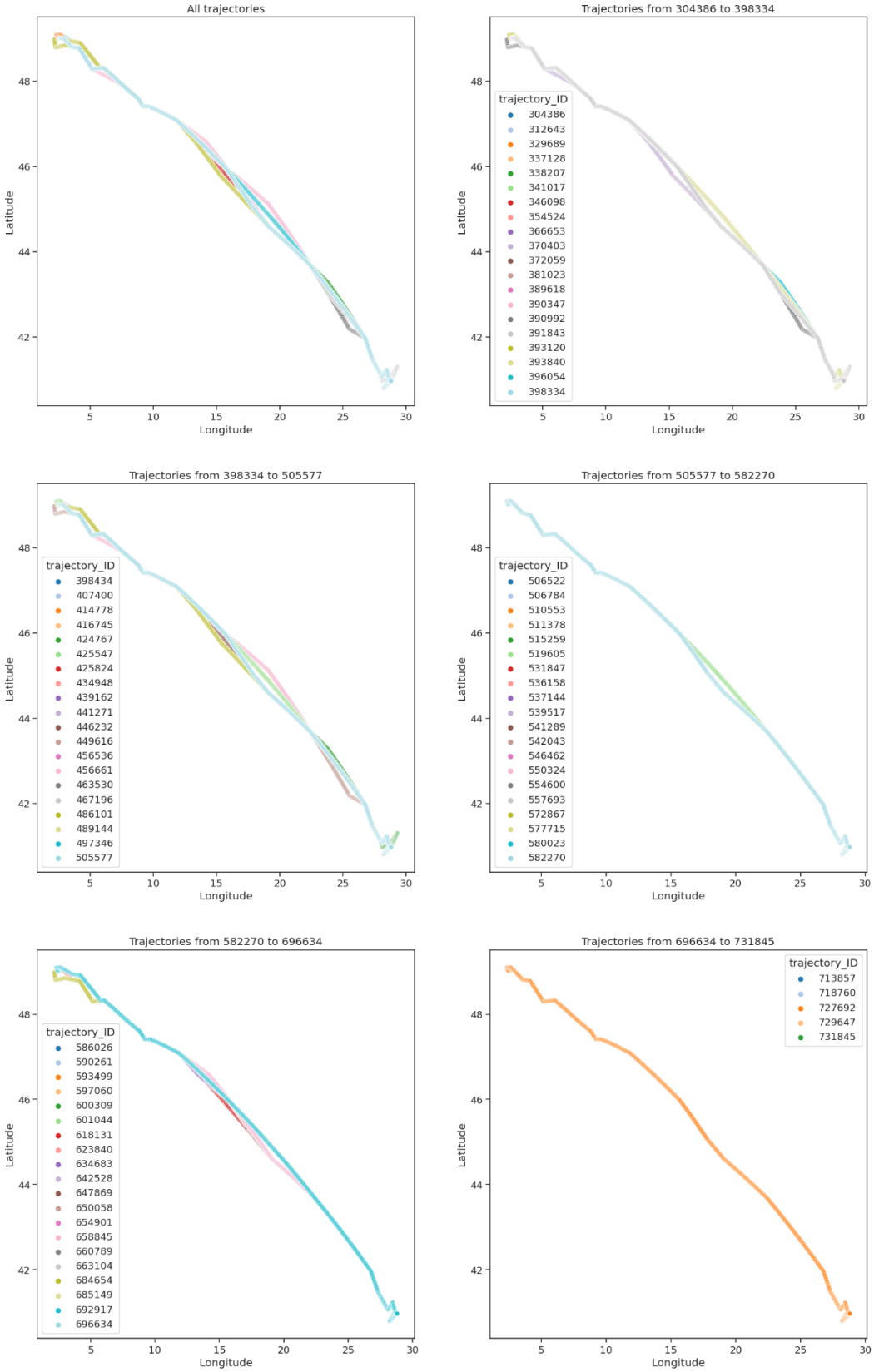

Figure 19: The 85 trajectories

3.1.4 Overview of the test dataset.

The test dataset contains 36,819 rows from 65 trajectories, and as in the training dataset they are distributed in six separate routes. The diagram in figure 20 illustrates the separate routes.

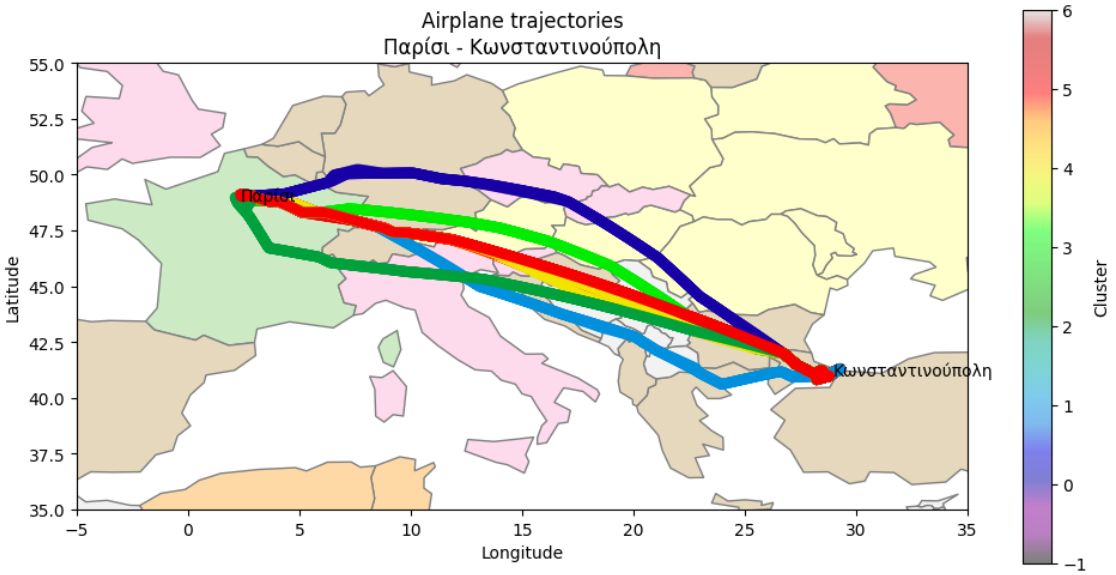

Figure 20: Airplane test trajectories.

The diagram in figure 21 illustrates the distribution of trajectories across the different routes. The route of cluster five (5) represents 59% of the whole dataset.

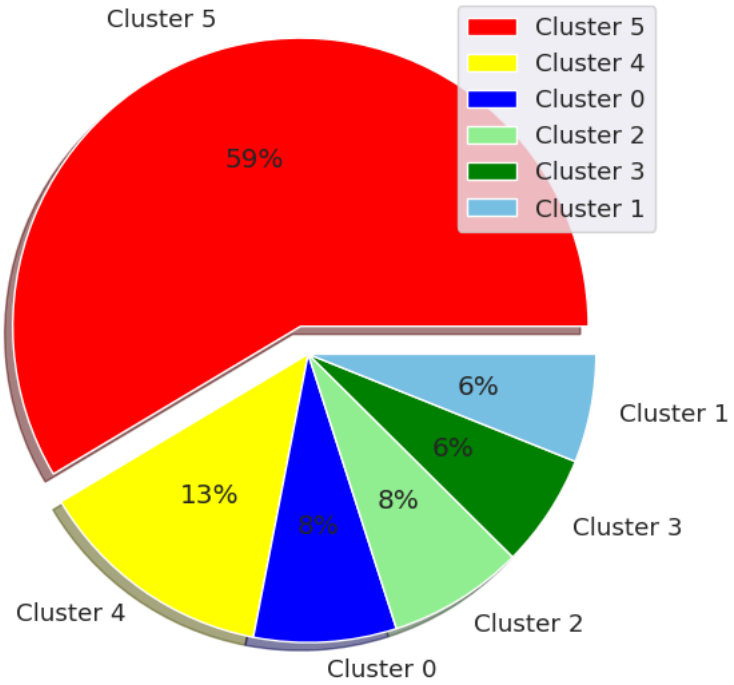

Figure 21: Distribution of test trajectories.

3.1.5 Cleaning test dataset

Here, the same pre-processing steps, as those in the training dataset described in section 3.1.2, do apply.

Overview of the test dataset after the data cleaning process.

After the data cleaning process, the test dataset contains 21,559 rows from 40 trajectories in route - data field “Cluster – five” (5). The diagram in figure 22 illustrates the altitude differences between the trajectories in route five (5).

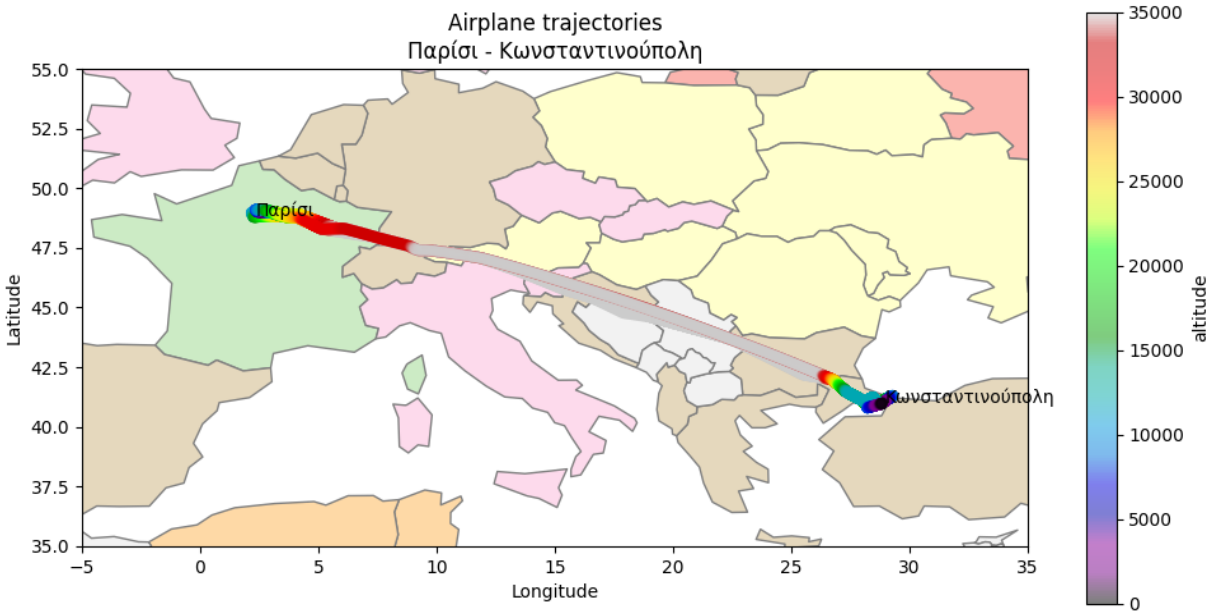

Figure 22: Test trajectories route 5 altitude.

Figure 23 illustrates trajectory ID cardinality in the test dataset.

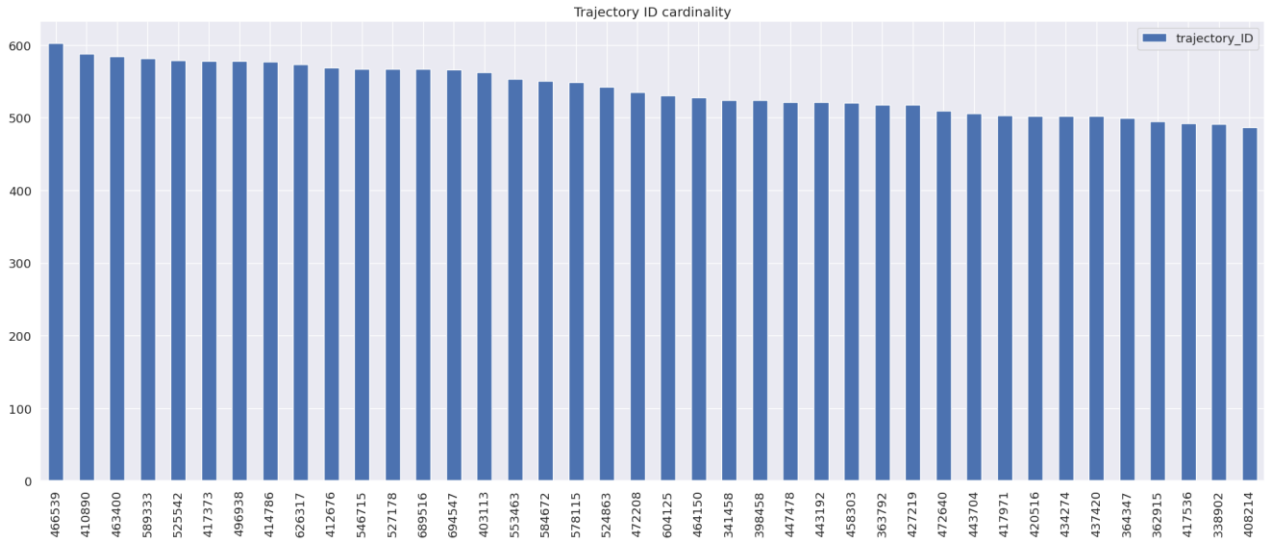

Figure 23: Trajectory ID cardinality in test dataset.

3.2 Implementation and training of RPL agent

It is challenging to train an intelligent agent to predict aircraft trajectories using unstructured traffic data due to the sparse reward issue. This section is dedicated to explaining the implementation and training of the Relay Policy Leaning (RPL) intelligent agent [1] as a potential solution for this long-horizon task.

Firstly, in the Relay Imitation Learning (RIL) first phase, unstructured aircraft trajectories are utilized as input to construct the high-level and low-level datasets. These datasets are used to train the agent with the high-level and low-level policies using a goal-conditioned Behavior Cloning algorithm. Finally, in the Relay Reinforcement Fine-tuning (RRF) second phase, a variant of the Trust Region Policy Optimization (TRPO) algorithm of Schulman et al. [4] employed to fine-tune the agent's training. The two-phase RPL algorithm is shown in Figure 24.

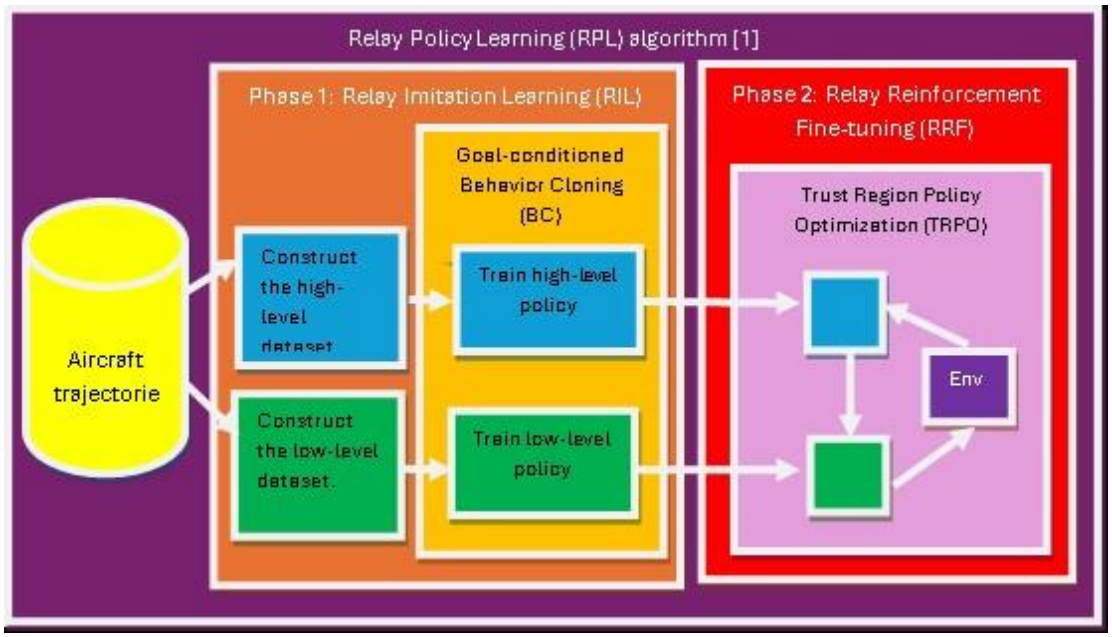

Figure 24: RPL intelligent agent [1]

The following Python libraries were employed in the implementation of the RPL intelligent agent.

NumPy

NumPy, short for Numerical Python, is an essential library for scientific computing. It provides multidimensional array objects and functions for working in domain of linear algebra, Fourier transform, and matrices [28].

Furthermore, tensors are essentially multidimensional arrays, which makes NumPy a central component in machine learning systems. Additionally, framework libraries TensorFlow and PyTorch utilize NumPy in core calculations.

Pandas

Pandas is a Python library built on the NumPy library and is frequently used in machine learning projects because has functions for analysing, cleaning, exploring, and manipulating data. Pandas can read various data formats, such as CSV files, and convert them into data frame objects. A data frame is a 2-dimensional data structure with rows and columns, like a spreadsheet [29]. Data frames are used for analysing and manipulating data in different ways.

The Panda library is utilized to the data preprocessing stage for reading, manipulating, and preserving the read Comma Separated Value (CSV) files of training and testing demonstrations datasets to DataFrames.

Geopandas

Geopandas is a Python library that provides support for geospatial data to Pandas DataFrames to make working with them more efficient [27].

The Geopandas library is utilized to the data preprocessing stage to plot airplane traffic data of the route from Paris to Istanbul. Additionally, it is used in the experiments results stage to plot the agent's generated trajectories.

Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python [40].

The Matplotlib library is utilized to the data preprocessing and evaluation stages to plot Pandas DataFrames.

Gymnasium

Gymnasium is a Python library for developing and evaluating reinforcement learning algorithms. It is a very versatile toolkit that is compatible with both TensorFlow and PyTorch libraries [30]. The library includes several pre-built environments for testing reinforcement learning agents, e.g., Atari games and robotics tasks. Arguably the most crucial feature of Gymnasium is that all environments share the same structure.

The Gymnasium library is utilized to provide the agent with a completely custom environment for testing Relay Policy Learning (RPL) algorithm.

PyTorch

PyTorch is an open-source machine learning (ML) framework based on the Python programming language and the Torch library. Torch is an open-source ML library used for creating deep neural networks and is written in the Lua scripting language [31].

The PyTorch library is used for the goal-conditioned Behaviour Cloning (BC) of low-level and high-level datasets, later outlined in detail.

Stable Baselines3

Stable Baselines3 (SB3) is a set of reliable implementations of reinforcement learning algorithms in PyTorch. It is the next major version of Stable Baselines [32]. SB3 is a deep reinforcement library that uses PyTorch for the backend and provides several implemented algorithms and features, for both online and offline reinforcement learning algorithms [41]. SB3 has implement experimental features in a separate SB3-Contrib library [42,43].

The SB3-Contrib library is utilized for Relay Reinforcement Fine-tuning (RRF) using the Trust Region Policy Optimization (TRPO) algorithm [44].

3.2.1 Relay Imitation Learning (RIL)

This section focuses on the phase of Relay Imitation Learning (RIL) implementation of the Relay Policy Learning (RPL) algorithm [1].

Relay data relabelling augmentation algorithm.

The relay data relabelling augmentation algorithm [1], described in section 2.4.2, allow us to learn goal-conditioning hierarchical policies without explicit labelling. In detail, we construct a low-level dataset D_l and a high-level dataset D_h by iterating through the unstructured traffic data of aircraft trajectories, which are included in the training dataset D, described in section 3.1.

Constructing the low-level dataset D_l

The low-level dataset D_l, described in Algorithm 2 in section 2.4.2 was constructed by iterating through the traffic data of aircraft trajectories included in the training dataset D. Firstly, we choose the low-level sliding window size W_l

to be 30, and then generate state-action-goal (s, a, s_g^l) tuples for low-level dataset D_l, within the sliding window W_l along the traffic data of aircraft trajectories.

After the data cleaning process in section 3.1.2, the aircraft trajectories, which are included in the training dataset D, are 85 trajectories in 46,467 rows. The low-level dataset created from training dataset has 85 trajectories in 1,347,543 rows. Figure 25 illustrates trajectory ID cardinality in low-level dataset.

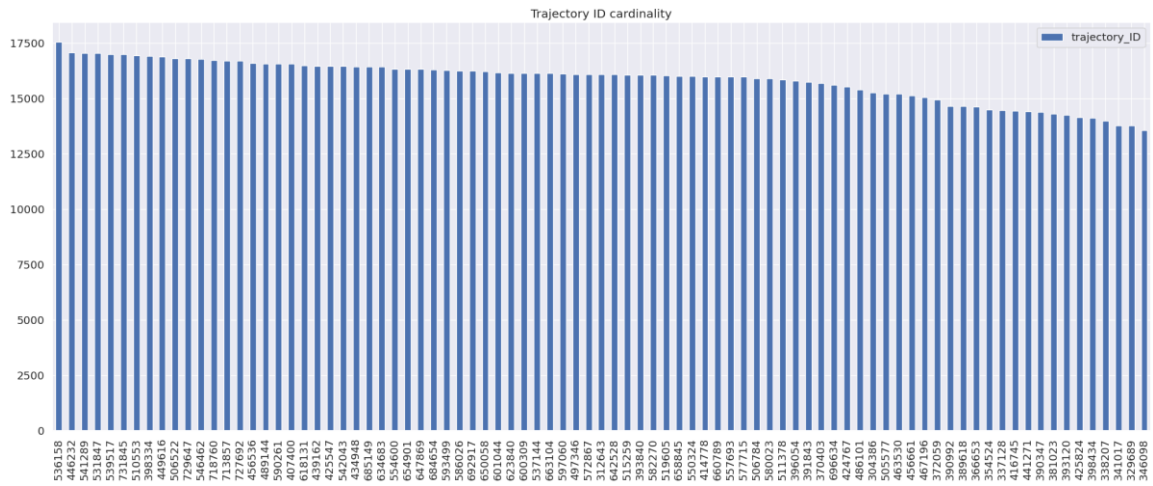

Figure 25: Trajectory ID cardinality in low-level dataset.

Constructing the high-level dataset D_h

Similarly, the high-level dataset D_h, described in Algorithm 3 in section 2.4.2 was constructed by iterating through the traffic data of aircraft trajectories included in the training dataset D. Firstly, we choose the high-level sliding window size W_h to be 260, and then generate state-action(subgoal)-goal tuples for high-level dataset D_h, within the sliding window W_h along the traffic data of aircraft trajectories.

After the data cleaning process in section 3.1.2, the traffic data of aircraft trajectories, which are included in the training dataset D, was 85 trajectories in 46,467 rows. The high-level dataset created from training dataset has 85 trajectories in 12,034,953 rows. Figure 26 illustrates trajectory ID cardinality in high-level dataset.

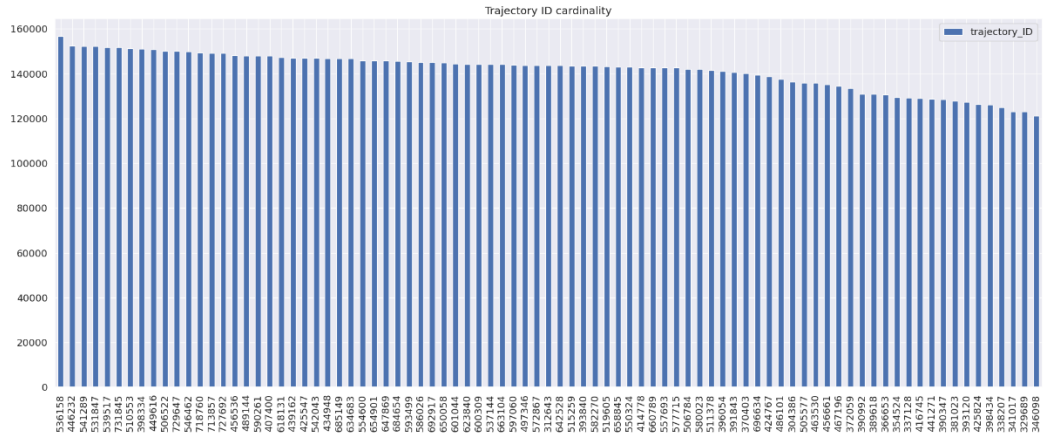

Figure 26: Trajectory ID cardinality in high-level dataset.

Goal-conditioned Behavior Cloning (BC)

Goal-conditioned Behavior Cloning (BC) is a simple imitation learning procedure that given the low-level and high-level datasets, trains a model for the high-level policy π_θ^h and for the low-level policy π_φ^l by maximizing the likelihood of the actions taken given the corresponding states and goals.

Deep Neural Network (DNN) Architecture

The Deep Neural Network (DNN) implementing the BC method was fed with airplane's longitude, latitude, and altitude. The goal is the DNN to be trained to predict the airplane coordinates for the next step. This is a multi-target Multilinear Regression Model (MRM), which is a machine learning model that utilizes multiple features as input to make multiple predictions with continuous values.

The final model architecture consisted of a Multi-Layer Perceptron (MLP) feedforward artificial neural network, which was fully connected to two hidden layers, as shown in Table 2.

Table 2: DNN architecture

Layer	Description
Feature Extractor	Flatten extractor
Input	Three values: longitude, latitude and altitude.
Linear	256 filters, RELU activation function
Linear	256 filters, RELU activation function
Output	Three values: longitude, latitude and altitude.

The network is separated into two mains sections. The first section comprises an extractor whose role is to extract features from observations. Coordinates are vector observations, so the features extractor is simply a Flatten Layer for each observation. The second section is a fully connected DNN with a nonlinear RELU activation function, which is used to train both the high-level policy π_{ϱ}^{h} and the low-level policy π_{φ}^{l} of the goal-conditioned BC.

All imitation learning algorithms are trained with the ADAM optimizer using a batch size of 128 and a learning rate of 0.005. The network's output is the predicted longitude, latitude, and altitude.

Coordinates Normalization

To enhance the Deep Neural Network's stability, the dataset features are normalized prior to being fed to it. Table 3 shows the airplane's longitude, latitude, and altitude zero-mean normalization formulas.

Table 3: Normalize formulas.

Normalize formulas
Normalize Longitude = Longitude – Longitude Average / Longitude Standard Deviation
Normalize Latitude = Latitude – Latitude Average / Latitude Standard Deviation
Normalize Altitude = Altitude – Altitude Average / Altitude Standard Deviation

Similarly, table 4 shows formulas utilized to unnormalize the trajectories generated by the trained airplane agent.

Table 4: Unnormalize formulas.

Unnormalize formulas
Unnormalize Longitude = Longitude * Longitude Standard Deviation + Longitude Average
Unnormalize Latitude = Latitude * Latitude Standard Deviation + Latitude Average
Unnormalize Altitude = Altitude * Altitude Standard Deviation + Altitude Average

3.2.2 Relay Reinforcement Fine-tuning (RRF)

This section focuses on the phase of Relay Reinforcement Fine-tuning (RRF) implementation of the Relay Policy Learning (RPL) algorithm.

Custom RL environment

To train the intelligence agent a custom environment created using the gymnasium interface [30], which includes all the necessary components to train the airplane agent. The agent was trained to cover the distance from Paris airport to Istanbul airport. This is a sparse binary reward task, in which the agent receives reward when arriving to the Istanbul airport. Additionally, the episode terminates in case of exceeding the 3,000-time steps. The observation space is a 3-dimensional continuous space, which observes the longitude, latitude, and altitude of the airplane agent. The action space is a 3-dimensional action space that corresponds to longitude, latitude, and altitude modifications to the airplane agent.

Trust Region Policy Optimization

In detail, as described in section 2.4.3 Gupta et al. [1] employ a goal-conditioned Hierarchical Reinforcement Learning (HRL) algorithm which is a variant of the Trust Region Policy Optimization (TRPO) algorithm proposed by Schulman and al. [4].

4 Experimental Results

This section provides the results from evaluating the agent using different configurations. Our experiments aim to answer the following questions:

1. Does Relay Imitation Learning (RIL) algorithm improve Reinforcement Learning Process to predict aircraft trajectories?
2. Is it possible to use Relay Policy Learning (RPL) algorithm to predict aircraft trajectories?

4.1 Performance metrics

For comparison, we trained four agents, the first agent trained exclusively with the Trust Region Policy Optimization (TRPO) algorithm. The second agent was trained solely with the low-level goal BC of the Relay Imitation Learning (RIL) algorithm. The third agent was trained solely with the high-level goal BC of the Relay Imitation Learning (RIL) algorithm. Finally, the fourth agent was trained with the Relay Policy Learning (RPL) algorithm. Afterwards, we employ the trained agents to predict five trajectories for the route Paris to Istanbul airport and plot the outcomes.

Furthermore, we compare the RPL agent's predicted trajectories to the original trajectories from Paris airport to Istanbul airport and plot the results.

4.2 Results

4.2.1 TRPO agent

The diagram in figure 27 illustrates the agent's five generated trajectories when it has been trained exclusively with Reinforcement Learning (RL) algorithm. Specifically, it was trained 1e5 time-steps with the Trust Region Policy Optimization algorithm, whose policy network was not pre-trained in any way. As we can observe, the agent cannot predict the root neither the correct altitude.

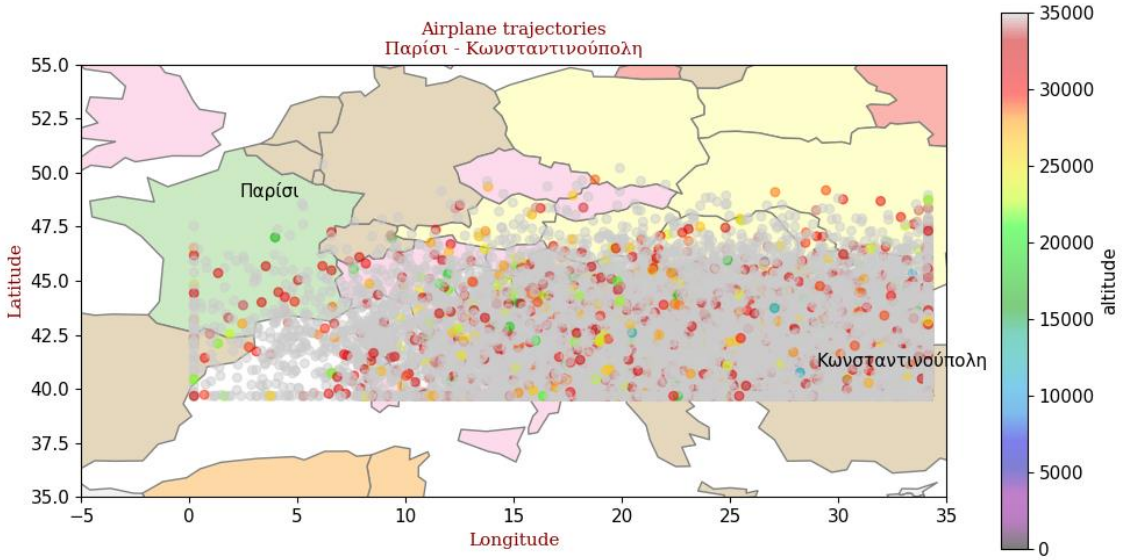

Figure 27: TRPO algorithm agent trained (1e5 time-steps).

The diagram in figure 28 illustrates the longitude and latitude of trajectories.

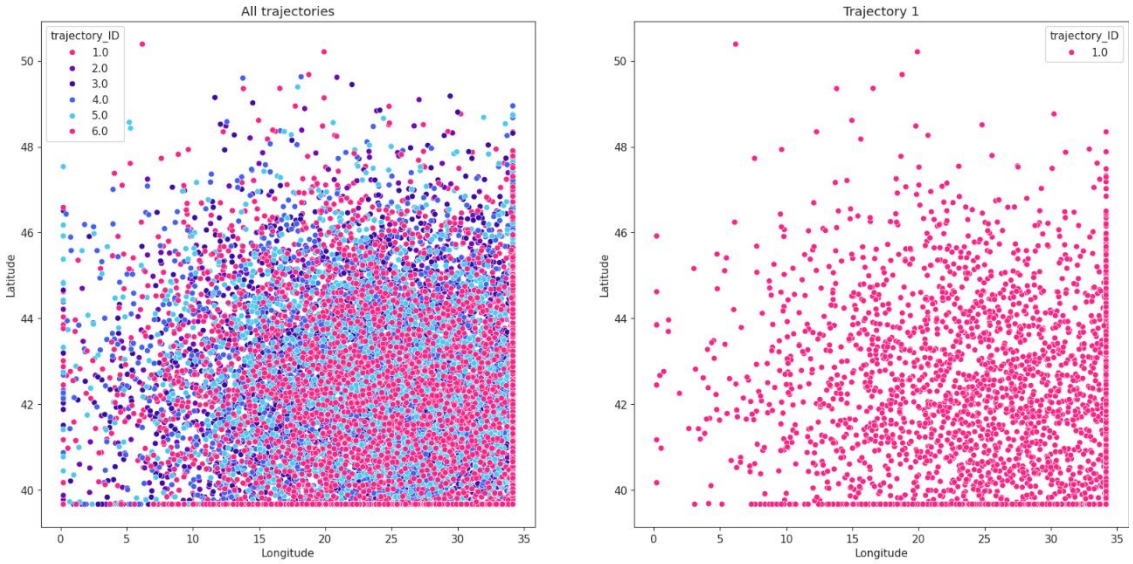

Figure 28: TRPO agent longitude and latitude trajectories (1e5 time-steps).

The diagram in figure 29 illustrates agent's five generated trajectories when enhancing the time-steps training to 1e6: It still does not provide valid predictions.

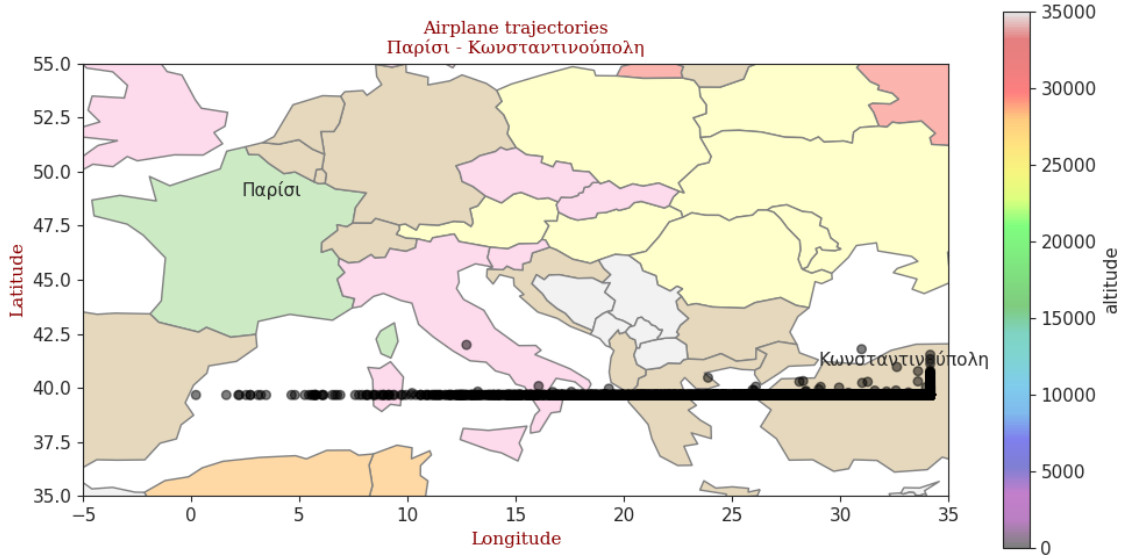

Figure 29: TRPO algorithm agent trained (1e6 time-steps).

The diagram in figure 30 illustrates the longitude and latitude of trajectories.

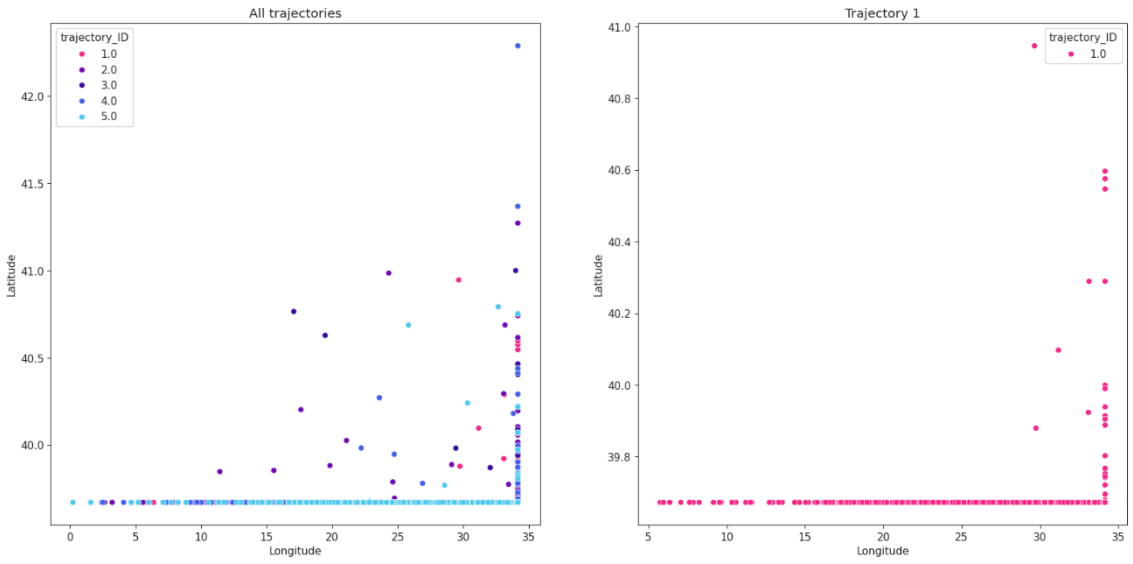

Figure 30: TRPO agent longitude and latitude trajectories (1e6 time-steps).

4.2.2 Goal-conditioned low-level BC agent

According to the Relay Imitation Learning (RIL) phase, the agent was first trained in 10 epochs with the goal conditioned low-level Behavior Cloning (BC) algorithm. The diagram in figure 31 illustrates agent's five generated trajectories. As we can observe, the agent was able to predict the correct root and the actual altitude, in a quite adequate way.

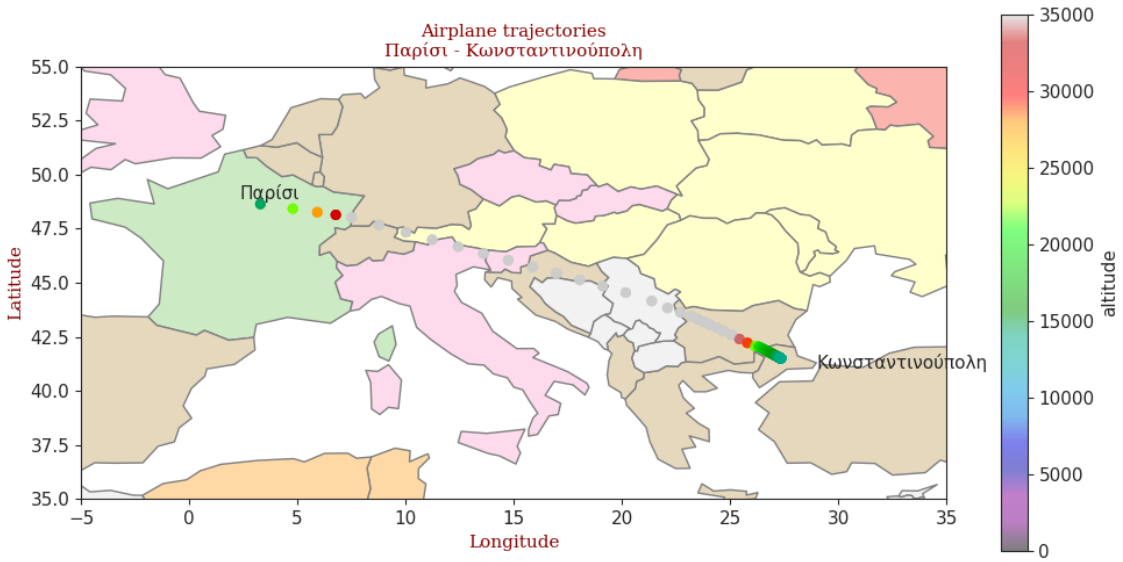

Figure 31: goal BC low-level algorithm agent trained (10 epochs).

The diagram in Figure 32 illustrates the longitude and latitude of trajectories.

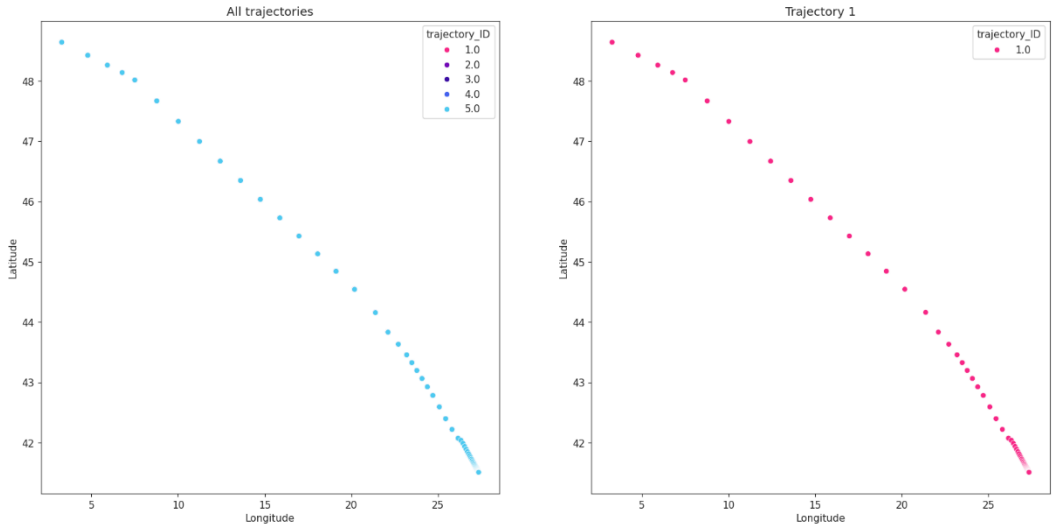

Figure 32: Goal BC low-level agent longitude and latitude trajectories (10 epochs). In the left figure, points for trajectories 1 to 4 coincide to the points of the 5th, depicted.

4.2.3 Goal-conditioned high-level BC agent

Then, the agent was trained 15 epochs with the high-level goal conditioned Behavior Cloning (BC) algorithm. The diagram in figure 33 illustrates agent's five generated trajectories. As we can observe, agent was able to predict several subgoals. However, all subgoals are from the middle of the route until Istanbul airport due to the high-level window size was set to 260 . Additionally, the altitude predictions are not correct.

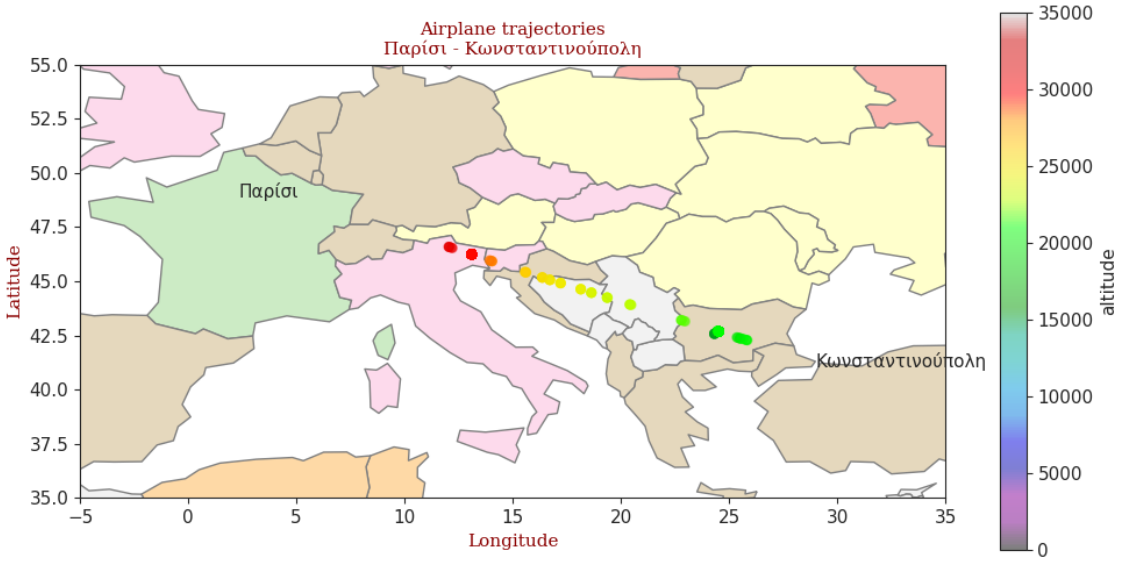

Figure 33: goal BC high-level algorithm agent trained (15 epochs).

The diagram in figure 34 illustrates the longitude and latitude of trajectories.

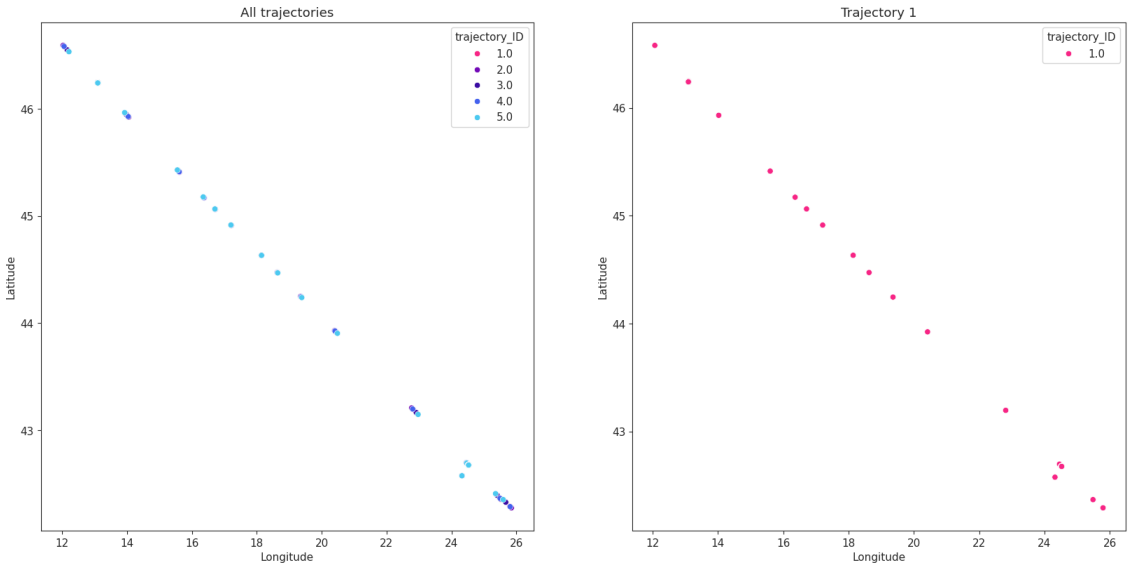

Figure 34: Goal BC high-level agent longitude and latitude trajectories (15 epochs). In the left figure, points for trajectories 1 to 4 coincide to the points of the 5th, depicted.

4.2.4 RPL agent

Finally, an agent was trained with the Relay Policy Learning (RPL) algorithm. The diagram in figure 35 illustrates the trained agent's generated trajectories. As we can observe, RL fine-tunning succeeds to improve RIL predictions, however there is still a high altitude predicted in Paris airport.

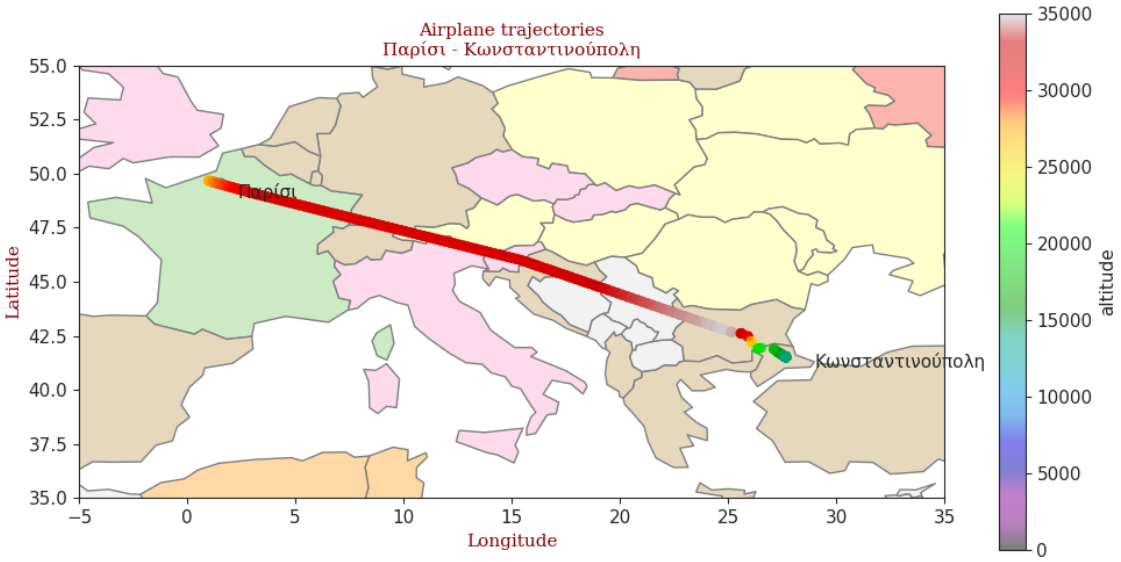

Figure 35: RPL trained agent (1e2 time-steps).

The diagram in figure 36 illustrates the longitude and latitude of trajectories.

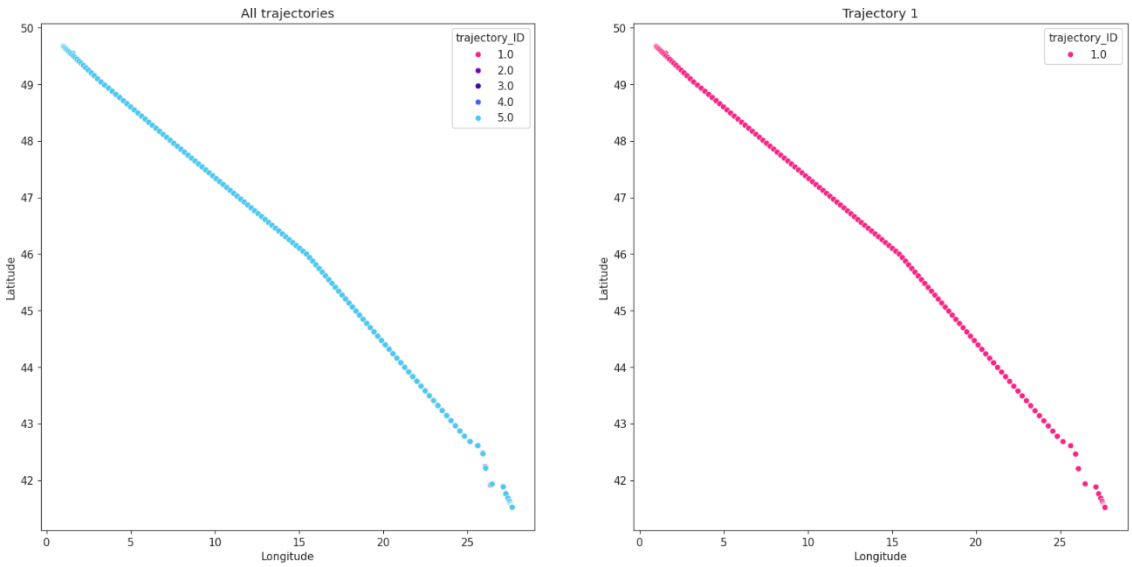

Figure 36: RPL agent longitude and latitude trajectories (1e2 time-steps). In the left figure, points for trajectories 1 to 4 coincide to the points of the 5th, depicted.

4.2.5 Compare original trajectories with RPL trajectories.

The diagram in figure 37 illustrates the comparison of an original trajectory with the predicted trajectory when the agent has been trained with the Relay Policy Learning (RPL) algorithm.

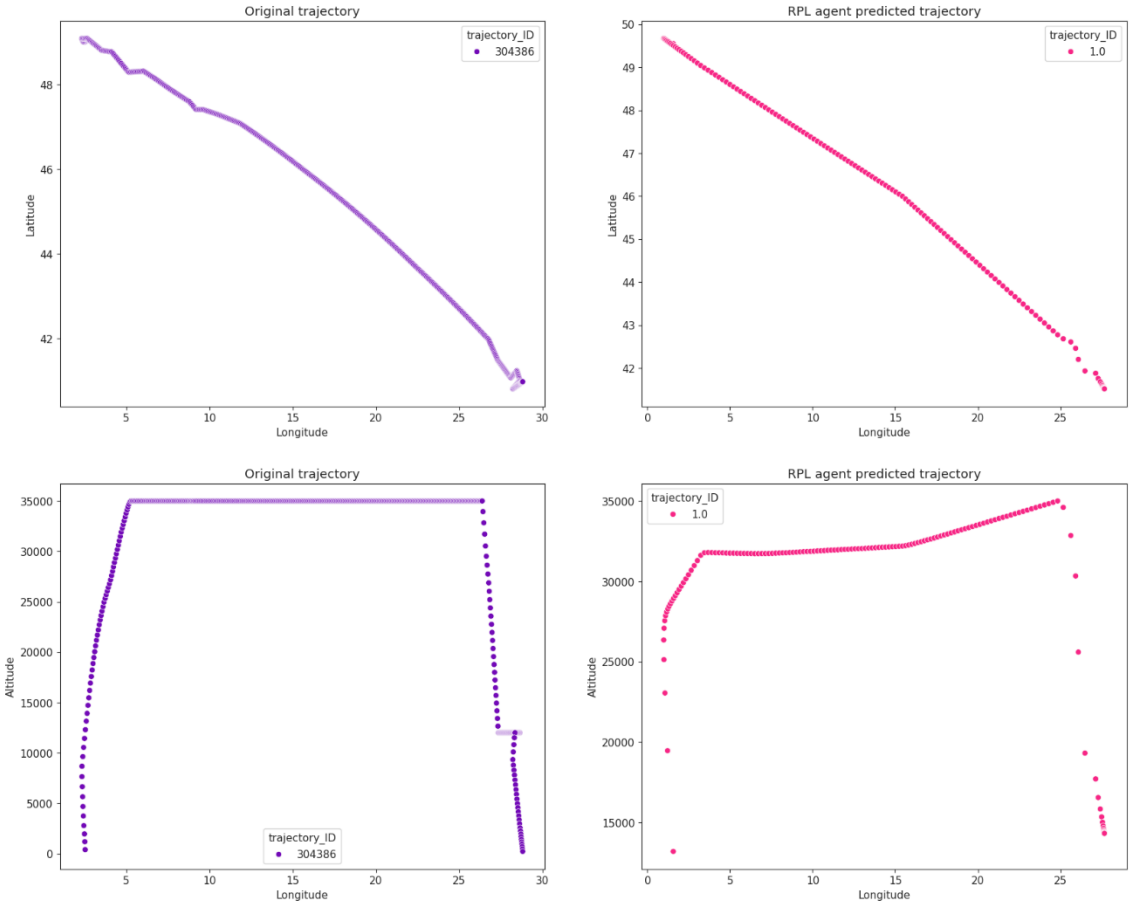

Figure 37: Visual comparison between the original and the RPL predicted trajectory.

As we can observe, predicted trajectory it is close to the original trajectory in longitude and latitude predictions. However, the altitude predictions are not accurate.

5 Conclusions and future work

In this thesis, Reinforcement Learning and Imitation Learning methods have been explored in the context of predicting long-horizon execution of tasks. In particular, the thesis focuses on the direct imitating supervised learning methods and the on-policy, model-free Reinforcement Learning methods, to model aircraft's trajectory prediction in the aviation domain.

Summarizing the work conducted in the context of this thesis we reengineer the Relay Policy Learning (RPL) algorithm [1] proposed by Gupta et al. and evaluate its use to predict aircraft trajectories. RPL is a two-phase hierarchical approach, consisting of a Relay Imitation Learning (RIL) phase that produces goal-conditioned hierarchical policies, and a Relay Reinforcement Fine-tuning (RRF) phase that fine-tunes these policies for task performance. It utilizes a dataset with long aircraft trajectories containing unstructured, unlabelled routes from Paris to Istanbul. First, the dataset was pre-processed to correct imperfections. Then, created low-level and high-level datasets through the relay-data relabelling augmentation of the RPL algorithm [1], which allow us to learn goal-conditioning hierarchical policies without explicit labelling. Afterwards, the created datasets are used to learn hierarchical Imitation Learning policies using a goal-conditioned Behavior Cloning method. Training Deep Neural Networks (DNNs) to predict airplane longitude, latitude, and altitude for the next step. To enhance the Deep Neural Network's stability, the datasets features are normalized prior to being fed to it. Finally, the two policies from the Relay Imitation Learning (RIL) phase are fine-tuning with Reinforcement Learning. Specifically, using the Trust Region Policy Optimization (TRPO) on-policy algorithm proposed by Schulman et al. [4]. To train the intelligence agent a custom environment created using the gymnasium interface.

Predicting aircrafts' trajectories can be challenging because it requires extensive exploration. The use of Imitation Learning to bootstrap the process of Reinforcement Learning, helps to overcome exploration challenges, while the RL fine-tuning allows the policy to improve based on actual task objective. The

results indicate that it is possible to use RPL algorithm in aircraft trajectories predictions. For comparison, we trained four agents, The first agent trained exclusively with Reinforcement Leaning cannot provide valid predictions. The second agent trained in 10 epochs with the goal conditioned low-level Behavior Cloning (BC) algorithm was able to predict the correct root and the actual altitude, in a quite adequate way. The third agent trained in 15 epochs with the high-level goal conditioned Behavior Cloning (BC) algorithm was able to predict several subgoals. However, all subgoals were from the middle of the route until Istanbul airport, and the altitude predictions were not correct. Finally, the fourth agent trained with the Relay Policy Learning (RPL) algorithm was able to improve RIL predictions. We demonstrated the effectiveness of RPL method on comparison of an original trajectory with the predicted trajectory when the agent has been trained with the Relay Policy Learning (RPL) algorithm.

In conclusion, during this research, it became clear that the Relay Policy Learning (RPL) algorithm [1] can be used to predict aircraft trajectories. Furthermore, it improves generalization since it is trained on many subgoals. The main advantage of the RPL algorithm is that it is simple and very general, in that it can be applied to any demonstrated data, including easy to provide unsegmented, and unstructured demonstrations of meaningful behaviours. Therefore, for future work, we propose to further explore with off-policy Reinforcement Learning methods.

Bibliography

[1] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In CoRL, 2019.

[2] https://www.cs.utexas.edu/~yukez/cs391r_fall2021/slides/pre_11-04_Harshit.pdf

[3] S. Ross, G. J. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In AISTATS 2011.

[4] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. Trust region policy optimization. CoRR, abs/1502.05477, 2015. URL http://arxiv.org/abs/1502.05477.

[5] A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. Discrete Event Dynamic Systems, 2003.

[6] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In RSS 2018.

[7] Y. Zhu, Z. Wang, J. Merel, A. A. Rusu, T. Erez, S. Cabi, S. Tunyasuvunakool, J. Kramár, R. Hadsell, N. de Freitas, and N. Heess. Reinforcement and imitation learning for diverse visuomotor skills. In RSS 2018.

[8] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel. Overcoming exploration in reinforcement learning with demonstrations. ICRA 2018.

[9] L. P. Kaelbling. Learning to achieve goals. In Proceedings of the 13th International Joint Conference on Artificial Intelligence. Chambéry, France, August 28 - September 3, 1993, pages 1094–1099, 1993.

[10] O. Nachum, S. Gu, H. Lee, and S. Levine. Data-efficient hierarchical reinforcement learning. In Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada., pages 3307–3317, 2018.

[11] https://github.com/aravindr93/mjrl

[12]https://web.stanford.edu/class/cs237b/pdfs/lecture/lecture_10111213.pdf

[13] R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. Artificial Intelligence, 112:181–211, 1999

[14] R. Parr. Hierarchical Control and Learning for Markov Decision Processes. PhD thesis, University of California, Berkeley, CA, 1998.

[15] R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. In Advances in Neural Information Processing Systems: Proceedings of the 1997 Conference, Cambridge, MA, 1998. MIT Press.

[16] T. G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. Journal of Artificial Intelligence Research, 13:227–303, 2000.

[17] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, 2015.

[18] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. Nature, 529 (7587):484–489, 2016.

[19] D. A. Pomerleau, “Alvinn: An autonomous land vehicle in a neural network,” NIPS, pp. 305–313, 1989.

[20] M. Bojarski et al., “End to End Learning for Self-Driving Cars,” arXiv preprint arXiv:1604.07316, 2016.

[21] A. Giusti et al., “A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots,” in IEEE Robotics and Automation Letters., 2015, pp. 2377–3766.

[22] J. Nakanishi et al., “Learning from demonstration and adaptation of biped locomotion,” in Robotics and Autonomous Systems, vol. 47, no. 2-3, 2004, pp. 79–91.

[23] M. Kalakrishnan et al., “Learning Locomotion over Rough Terrain using Terrain Templates,” in The 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009.

[24] B. D. Ziebart et al., “Maximum Entropy Inverse Reinforcement Learning.” in AAAI Conference on Artificial Intelligence, 2008, pp. 1433–1438.

[25] P. Abbeel and A. Y. Ng, “Apprenticeship learning via inverse reinforcement learning,” in ICML, 2004, p. 1.

[26] C. Finn, S. Levine, and P. Abbeel, “Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization,” in ICML, 2016.

[27] "Geopandas", Geopandas.org. [Online]. Available: https://geopandas.org/en/stable/ [Accessed: 23 January 2024].

[28] "NumPy", Numpy.org. [Online]. Available: https://numpy.org/. [Accessed: 23 January 2024].

[29] "pandas - Python Data Analysis Library", Pandas.pydata.org, 2021. [Online]. Available: https://pandas.pydata.org/. [Accessed: 23 January 2024].

[30] "Gym: A toolkit for developing and comparing reinforcement learning.” https://gymnasium.farama.org/index.html [Accessed: 23 January 2024].

[31] https://pytorch.org/. [Accessed: 23 January 2024].

[32] Antonin Raffin et al., "Stable-Baselines3: Reliable Reinforcement Learning Implementations ", Journal of Machine Learning Research, 2021. [Online]. Available: https://jmlr.org/papers/volume22/20-1364/20-1364.pdf [Accessed: 25 January 2024].

[33] C. Lynch, M. Khansari, T. Xiao, V. Kumar, J. Tompson, S. Levine, and P. Sermanet. Learning latent plans from play. CoRR, abs/1903.01973, 2019.

[34] https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

[35] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: nature 521.7553 (2015), pp. 436–444.

[36] https://doi.org/10.48550/arXiv.1707.06203

[37] https://doi.org/10.48550/arXiv.1709.03153

[38] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba. Hindsight experience replay. CoRR, abs/1707.01495, 2017. URL http://arxiv.org/abs/1707.01495.

[39] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.corr.html

[40] https://matplotlib.org/ [Accessed: 23 January 2024]

[41] https://stable-baselines3.readthedocs.io/en/master/ [Accessed: 25 January 2024]

[42] https://sb3-contrib.readthedocs.io/en/master/ [Accessed: 25 January 2024]

[43] Raffin, Antonin and Hill, Ashley and Ernestus, Maximilian and Gleave, Adam and Kanervisto, Anssi and Dormann, Noah. Stable Baselines3, 2019. URL https://github.com/DLR-RM/stable-baselines3.

[44] https://sb3-contrib.readthedocs.io/en/master/modules/trpo.html# [Accessed: 25 January 2024]