DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

DLR

Applying Reinforcement Learning methods to flight trajectory planning on a network scale

Bachelor's thesis

for the degree

Bachelor of Science

of the course Computer Science with focus on Artificial Intelligence

at the Baden-Wuerttemberg Cooperative State University Mannheim

by

Fabian Schwarz

Submission on 25.08.2026

Processing Period:	01.06.2026 - 25.08.2026
Student id, course:	8971770, TINF23IKI1
Department:	Institute of Air Transport (LV)
Apprenticing company:	German Aerospace Center (DLR)
Company's supervisor:	M.Sc Julian Solzer
University's reviewer:	Prof. Dr. Alexander Dück

Declaration

I hereby assure you that I have written my bachelor's thesis on the

Subject

Applying Reinforcement Learning methods to flight trajectory planning on a network scale

independently and that I have not used any other sources and aids than those indicated.

I also assure you that the electronic version submitted is the same as the printed version.*

* if both versions are required.

Cologne, 25.08.2026

Abstract

Climate-aware aircraft trajectory planning has the potential to significantly reduce the impact of aviation on the Earth's climate. However, much of the existing research focuses on optimizing individual flight trajectories, which can result in solutions that are difficult to integrate into the wider air traffic network. Multi-agent reinforcement learning (MARL) offers a promising approach to jointly consider climate impact and operational feasibility at the network level, but requires suitable environments and data-generation pipelines to support efficient training and experimentation.

This thesis designs and implements a system to support future research into climate-aware trajectory planning using MARL. Based on existing work, a two-stage architecture is adopted. In the first stage, individual trajectories are optimized with respect to their climate impact. In the second stage, a MARL system modifies these trajectories to balance climate impact against air traffic complexity. The system comprises a data-generation pipeline, a high-performance training environment, and the MARL-based balancing component, with particular focus placed on efficient parallel simulation of large numbers of aircraft agents.

Initial experiments using the proposed system demonstrate its ability to integrate the individual trajectory optimization and network-level balancing stages. At the same time, the experiments reveal challenges in the current approach, particularly regarding the reward formulation and the emergence of degenerate policies. These results provide directions for further development of the reward structure and MARL system and establish a foundation for future research into network-scale climate-aware trajectory planning.

Contents

List of Figures VI

List of Tables VII

List of abbreviations VIII

1. Introduction 1
- 1.1. Motivation . . . 1
- 1.2. Aims and objectives . . . 3
- 1.3. Structure . . . 4

2. Fundamentals 5
- 2.1. Reinforcement Learning . . . 5
  - 2.1.1. Trust Region Policy Optimization . . . 10
  - 2.1.2. Proximal Policy Optimization . . . 11
- 2.2. Multi Agent Reinforcement Learning . . . 13
  - 2.2.1. Multi Agent PPO . . . 15
- 2.3. Polars . . . 16

3. System Architecture 17
- 3.1. High level concept . . . 17
- 3.2. Baseline Trajectory Structure . . . 18
- 3.3. Multi Agent Reinforcement Learning (MARL) system design . . . 20

4. Implementation 22
- 4.1. Baseline Generation . . . 22
  - 4.1.1. Loading Data . . . 23
  - 4.1.2. Traffic data preprocessing . . . 24
  - 4.1.3. Generating micro-scale climate-optimal trajectories . . . 26
  - 4.1.4. Altitude and speed adjustments . . . 27
- 4.2. Environment . . . 31
  - 4.2.1. State . . . 31
  - 4.2.2. State update . . . 32

4.2.3. Observation 34
4.2.4. Reward calculation 35
4.3. Multi Agent Reinforcement Learning (MARL) system 39
4.3.1. Actor network 39
4.3.2. Critic network 39
4.3.3. Rollout Buffer 42
4.3.4. Training Loop 42

5. Results and discussion 45
5.1. Evaluation Policies 45
5.2. Initial results 46
5.3. Speed Penalty 52
5.3.1. Penalty Implementation 52
5.3.2. Results 54

6. Conclusion and future research topics 56
6.1. Conclusion 56
6.2. Future research topics 58

Bibliography 59

List of Figures

2.1. Reinforcement Learning (RL) feedback loop [15] 6

3.1. Trajectory with pre-computed decision points. Own illustration; aircraft icon from [32] 18
3.2. Trajectory with dense baseline points and decision point approximation. Own illustration; aircraft icon from [32] 19

4.1. Baseline trajectory generation 23
4.2. Altitude profile generated by pyTOM 27
4.3. Route generated by pyTOM (red) vs the shortest route (black, dotted) 28
4.4. Altitude adjustments made by the trajectory rebuilder 29
4.5. Original speed trajectory 30
4.6. Adjusted speed trajectory 30
4.7. Actor Network. Visualization using [42] 40
4.8. Critic Network. Visualization using [42] 41

5.1. Cumulative evaluation climate (left) and complexity (right) rewards throughout training 47
5.2. Entropy throughout training 47
5.3. Mean Balanced Complexity (left) and Climate (right) Training Rewards throughout training 48
5.4. Correlation between Mean Agent Speed (orange) and Complexity (left) Climate Reward (right) during training with a fixed environment 50
5.5. Evaluation mean action probability throughout training 51
5.6. Evaluation mean action throughout training 51
5.7. Evaluation mean action throughout training 54
5.8. Policy Loss (left) and Value Loss (right) throughout training 55

A.1. Distribution of active flights vs complexity 65
A.2. Correlation between number of active flights and complexity 65
A.3. Power law fitting to find β 66
A.4. Pairwise EMD Comparison by Max Active Flights 67
A.5. Spatial distribution of training datasets 69

List of Tables

5.1. Mean, cumulative, minimum, and maximum evaluation metrics across policies 49

A.1. Number of baseline trajectories and mean baseline length for each dataset. 68
A.2. State dataframe 70
A.3. Multi Agent PPO (MAPPO)-specific Parameters 71

List of abbreviations

AC Actor Critic

aCCF algorithmic Climate Change Function

BADA Base of Aircraft Data

CARA Computer for Advanced Research in Aerospace

CO_2 Carbon Dioxide

CTCE Centralized Training Centralized Execution

CTDE Centralized Training Decentralized Execution

DDR2 Demand Data Repository 2

Dec-POMDP Decentralized partially observable Markov Decision Process

DLR German Aerospace Center

DQN Deep Q-Network

DSL Domain Specific Language

ECMWF European Centre for Medium-Range Weather Forecasts

EMD Earth Mover's Distance

ERA ECMWF Re-Analysis

GAE Generalized Advantage Estimation

HPC High Performance Computing

IL Independant Learning

KL-divergence Kullback-Leibler divergence

LV Institute for Air Transport

MAE Mean Average Error

MAPPO Multi Agent PPO

MARL Multi Agent Reinforcement Learning

MDP Markov Decision Process

ML Machine Learning

MSE Mean Squared Error

NO_x Nitrogen Oxides

NOOP no operation

NPGP Next-Point Greedy Policy

POMDP Partially observable Markov Decision Process

PPO Proximal Policy Optimization

PyTOM Python Trajectory Optimization Module

ReLU Rectified Linear Unit

RL Reinforcement Learning

SAF Sustainable Aviation Fuel

TRPO Trust Region Policy Optimization

UAV Unmanned Aerial Vehicle

1. Introduction

1.1. Motivation

The impacts of climate change are reaching unprecedented levels, with global mean surface temperatures exceeding 1.1°C above pre-industrial averages in the last decade [1, 2]. This warming trend has triggered profound systemic shifts, as evidenced by the escalating frequency and severity of climate-related disasters, which now regularly impose multi-billion-dollar economic burdens and threaten livelihoods worldwide [2, 3].

Projecting under current emission trajectories, these impacts are set to intensify further, jeopardizing both ecological stability and global socioeconomic systems [1]. To counter this trajectory, international frameworks such as the Paris Agreement and corresponding EU climate legislation have established binding targets to limit warming to 2.0°C and achieve net-zero emissions by mid-century.

With transport accounting for approximately 15% of global greenhouse gas emissions, and its emissions having grown faster than any other end-use sector since 2010 (+1.8% per year) [4], the sector is lagging behind broader decarbonization efforts. The emissions of the aviation sector account for approximately 4% of global anthropogenic climate forcing and continue to grow rapidly [4, 5].

European Air traffic reached 99% of 2019 pre-covid levels in 2026 and is expected to continue growth, reaching an estimated 2 billion passengers in 2050 [6, 7]. Global air traffic is projected to grow at a rate of 3.6% annually and reach 13 billion passengers by 2050 [7]. These projections highlight the importance of taking measures to decrease aviation's contribution to global warming.

The climate impact of aviation arises from direct emissions of Carbon Dioxide (CO_2) as well as from *non-CO_2-effects*. Direct CO_2 emissions make up one third of total climate forcing of aviation with the remaining two thirds of being made up by non CO_2-effects [5]. NonCO_2-effects include Nitrogen Oxides (NO_x) emissions along with water vapor and soot that can result in contrail formation that can have heating or cooling effects based on a wide range of factors. Unlike direct CO_2 emissions, non-CO_2 effects are influenced not only by the quantity of emission but also the emission location, time and meteorological boundary conditions [5].

To reduce the impact aviation has on the climate, different measures are being explored. This includes the development of hydrogen as well as electric aircraft to directly reduce emissions [8]. Sustainable Aviation Fuels (SAFs) are also deployed to reduce non CO_2-effects. Another approach is the optimization of aircraft trajectories in regards to their climate impact. By avoiding geographical areas and altitudes that result in a heating effect and favoring areas that can be regarded as neutral or even offer the potential for cooling contrails, the climate impact of aviation can be reduced quickly and efficiently. Smith et al. estimate that contrail avoidance through optimized routes could reduce global warming by as much as 0.04K which equals 9% of the of the global temperature budget that remains within the Paris Agreement [9].

Previous work often focuses on optimizing individual routes in regard to their environmental impact. Yamashita et al. employ a multi-objective genetic algorithm to optimize North Atlantic flight trajectories, explicitly evaluating the trade-offs between operating costs, fuel efficiency, and climate effects [10]. Similarily Simorgh et al. develop an robust open source library for optimizing individual trajectories in the structured airspace utilizing GPUs and taking into account uncertainties in weather [11].

While optimizing individual routes can be seen as a natural starting point in decreasing air traffic's contribution to global warming, this approach does not effectively scale due to the emergence of similar flight paths which can lead to unsafe air traffic operations [12, 13]. Solving this issue presents itself as a nontrivial challenge as finding an optimal solution becomes increasingly difficult with a growing number of aircraft.

Recent studies by Baneshi et al. have demonstrated the effectiveness of Multi Agent Reinforcement Learning (MARL) to aid in the solution of this scaling problem through coordinated trajectory planning. Their initial research investigates coordinated trajectory planning for traffic conflict mitigation, while follow-up work employs Proximal Policy Optimization (PPO) to balance airspace complexity and climate impact mitigation in structured environments [12, 13].

The results of these studies demonstrate the potential of MARL for coordinating aircraft in climate-aware trajectory optimization on a network scale. At the same time, the use of MARL for this purpose is still an emerging field, leaving substantial room for further investigation. The combination of the increasing need for climate-aware aviation and the promising capabilities of MARL therefore provides the motivation for this work.

1.2. Aims and objectives

Building on the foundations of existing concepts, this thesis shifts focus to the engineering challenge of implementing a complete, reproducible reinforcement learning infrastructure for climate-aware route optimization from scratch. The primary aim is to design and validate a flexible system architecture that provides a stable, well-documented foundation for subsequent climate-aware routing research based on MARL.

To achieve this aim, the thesis pursues the following objectives:

1. Design an architecure that models the problem scope while maintaining a high degree of realism
2. Implementing a data pipeline to generate required training data for the training process
3. Implementing a performant and flexible environment that can facilitate training of different policies

4. Propose a initial architecture for a MARL system to learn policies in the environment
5. Conduct initial feasibility experiments using the MARL system to validate the environment and highlight challenges
6. Outline areas of future research

1.3. Structure

The thesis is structured as follows: After outlining fundamental technical concepts and foundations in section 2 the planned system architecture is introduced in section 3. Section 4 details the technical implementation of the architecture discussed in the previous chapter. This includes both supporting elements such as data preprocessing (section 4.1) as well as the environment (section 4.2) and the MARL system itself (section 4.3). Chapter 5 analyzes the results gathered from initial experiments conducted using the system and analyzes potential issues and solutions. Finally, chapter 6 summarizes the work and discusses further areas of research.

2. Fundamentals

This chapter outlines the theoretical background necessary for the development of the proposed MARL framework. The remainder of this chapter is organized as follows:

First, single-agent reinforcement learning is introduced and formalized. Subsequently, key concepts and algorithms specific to multi-agent reinforcement learning are detailed. Finally, the data types and sources required for the proposed method are described.

Please note that a unified notation system mostly inspired by Rashid et al. (2018) has been adopted throughout this work; consequently, some symbols may differ from those found in the cited literature. This adjustment ensures consistency between the single-agent and multi-agent sections as well as the remainder of the work.

2.1. Reinforcement Learning

Reinforcement Learning (RL) is an area of Machine Learning (ML) that deals with the behaviour of agents in defined environments. Agents take actions and try to maximize a reward they receive. This process is depicted in figure 2.1.

At every timestep t the agent a is in a state $s_t \in \mathcal{S}$ and picks an action $u_t \in \mathcal{U}$. The environment transitions into state s_{t+1} and the agent receives a reward r_{t+1}. In most real-world scenarios agents do not pick actions based on the real state but an observation z_t that is derived from the true state s using the observation function O:

$$z_t = O(s_t).$$

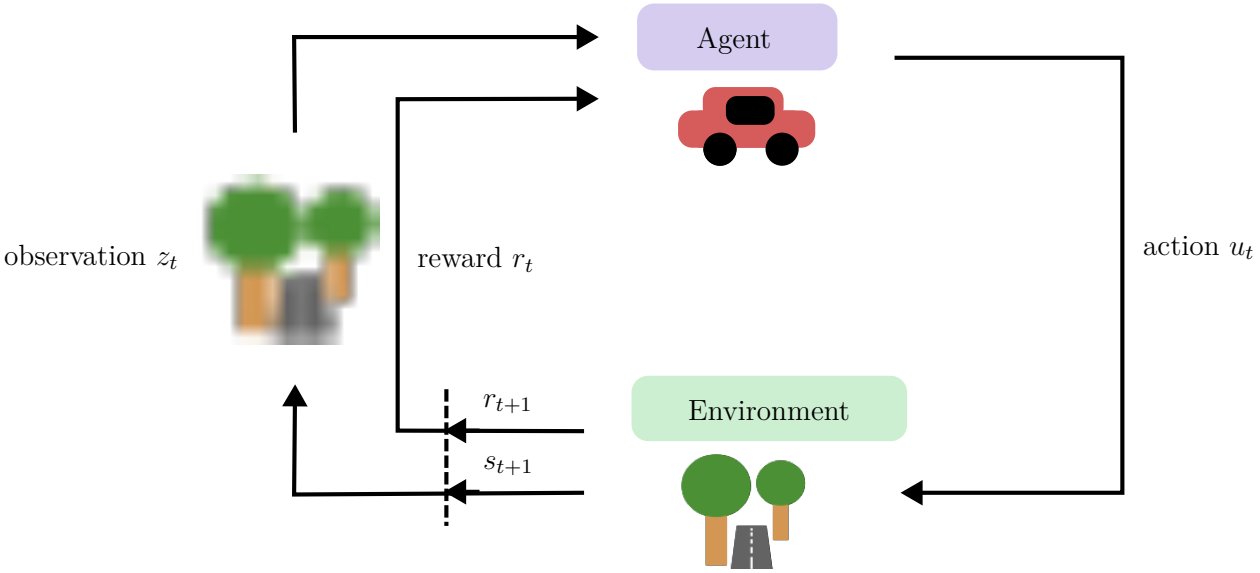


Figure 2.1.: RL feedback loop
[15]

In most literature, single agent RL problems are formally defined by an Markov Decision Process (MDP):

$$\mathcal{M} = (\mathcal{S}, \mathcal{U}, P, R, \gamma),$$

where $\mathcal{S}$ denotes the state space, $\mathcal{U}$ the action space, $P(s' \mid s, u)$ the state transition probability, $R(s, u)$ denotes the reward function and $\gamma \in [0, 1)$ denotes the discount factor [16, 17]. As described above, in most real-world decision making processes agents do not have access to the real state and are provided with an observation instead. These processes are referred to as a Partially observable Markov Decision Process (POMDP)

$$\mathcal{P} = (\mathcal{S}, \mathcal{U}, \mathcal{Z}, P, O, R, \gamma),$$

where $\mathcal{Z}$ is the observation space and O the observation function [16].
The *policy* used by agents to pick an action is denoted as $\pi(u \mid z)$ It maps observations to an action distribution:

$$\pi(u \mid z) = \Pr(u_t = u \mid z_t = z)$$

The goal of an agent is to maximize the discounted reward

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

where γ is the discount factor that controls the tradeoff between short-term reward and long-term decision making.

There are multiple algorithms and strategies to derive an optimal policy that can be broadly classified into three categories:

1. Value based reinforcement learning
2. Policy gradient reinforcement learning
3. Actor-Critic methods

Value-based methods

Value based methods derive a policy by iteratively approximating the state-value function $V^\pi(s)$ or the action-value function $Q^\pi(s, a)$. These functions estimate the expected cumulative discounted return that an agent can achieve by following policy π from a given state or state–action pair. The state-value function is formally defined as

$$V^\pi(s) = \mathbb{E}_\pi[G_t \mid s_t = s], \tag{2.1}$$

The action-value function extends this concept by conditioning on both the current state (or observation) and the executed action:

$$Q^\pi(s, u) = \mathbb{E}_\pi[G_t \mid s_t = s, U_t = u]. \tag{2.2}$$

Once a reliable estimate of $Q^\pi(s, u)$ is obtained, an optimal deterministic policy can be derived greedily by selecting the action that maximizes the estimated action-value:

$$\pi^*(s) = \arg\max_{u \in \mathcal{U}} Q^*(s, u). \tag{2.3}$$

During the training phase, actions are typically sampled stochastically (e.g., via an ε-greedy strategy) to encourage exploration of the state space. Conversely, at evaluation or deployment time, actions are selected deterministically according to the greedy policy to maximize performance [16].

In practical reinforcement learning scenarios, the number of possible states often far exceeds the memory and computational limits of tabular methods, rendering exact representations of V and Q infeasible. To address this, function approximation techniques - most notably deep neural networks—are employed to learn compact, generalizable representations:

$$\hat{v}(s; \mathbf{w}_v) \approx V_\pi(s); \quad \hat{q}(s, u; \mathbf{w}_q) \approx Q_\pi(s, u), \tag{2.4}$$

where $\mathbf{w}_v$ and $\mathbf{w}_q$ denote the learnable parameters of the respective value networks. Building on this principle, the Deep Q-Network (DQN) algorithm combines deep neural networks with temporal difference learning to approximate the optimal action-value function [18, 19]. A core advantage of value-based methods like Deep Q-Network (DQN) is their ability to perform off-policy learning. By storing past transitions in a replay buffer, the agent can learn from arbitrary state-action pairs, even when the data was collected under a different, outdated policy. This not only breaks temporal correlations between samples but also significantly improves sample efficiency and training stability [16].

Policy gradient methods

While value-based methods optimize auxiliary value functions and derive policies implicitly (e.g., via greedy strategies), policy gradient approaches parameterize the policy $\pi_\theta(s, u)$ directly. The objective is to maximize a performance measure $J(\theta)$, defined as the expected cumulative return:

$$J(\theta) = \mathbb{E}_{\pi_\theta}[G_t]$$

Optimization is performed via gradient ascent on this objective function. The theoretical basis for this update is the Policy Gradient Theorem, which defines the

gradient of the performance measure with respect to the parameters:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim d^\pi, u\sim\pi_\theta} \left[\nabla_\theta \log \pi_\theta(u \mid s) \cdot Q^\pi(s,u)\right], \tag{2.5}$$

where $d^\pi(s)$ is the discounted state visitation distribution and $Q^\pi(s,u)$ represents the action-value function under policy π. Mathematically, this implies that parameter updates should reinforce actions with high expected returns while penalizing those with low returns. A primary advantage of policy-based methods is their capacity to learn stochastic policies and handle continuous action spaces where discrete maximization would be intractable [16].

However, standard policy gradient implementations face significant challenges regarding convergence stability. Since the true action-value function $Q^\pi(s,u)$ is unknown, it is typically approximated using empirical Monte Carlo returns G_t. This substitution introduces high variance into the gradient estimates, as single samples of G_t are noisy and fail to distinguish between 'lucky' actions versus actions that were genuinely effective. Consequently, policy gradients often suffer from slow convergence and instability during training.

Actor Critic

AC methods address the high variance of standard policy estimation by explicitly modeling the action-value function. The architecture consists of two components: an *Actor* and a *Critic*. The actor maintains the parametrized policy $\pi_\theta(u|s)$ and determines actions, while the critic evaluates the current state or state-action pair to provide a learning signal.

Specifically, the critic approximates the value function $V^\pi(s)$ or $Q^\pi(s,u)$ using a separate network. By incorporating this estimate, the gradient estimator can be reformulated using the advantage function $A^\pi(s,u)$:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s,u} \left[\nabla_\theta \log \pi_\theta(u \mid s) \cdot A^\pi(s,u)\right]. \tag{2.6}$$

The advantage function is defined as the difference between the estimated action-value

and the state-value:

$$A(s,u) = Q(s,u) - V(s). \tag{2.7}$$

This formulation utilizes $V(s)$ as a baseline to center the gradient signal, thereby significantly reducing variance compared to using raw returns. The advantage function indicates how much better a specific action is compared to the average action expected by the current policy, allowing for more stable and efficient learning.

2.1.1. Trust Region Policy Optimization

Trust Region Policy Optimization (TRPO) improves upon standard policy gradient methods by ensuring stable and monotonic performance updates during training. A major challenge with regular policy gradient algorithms is that they typically require very low step sizes so as not to destabilize the optimization process. Such conservative updates result in poor sample efficiency, as large amounts of trajectory data are required to achieve meaningful progress. TRPO addresses this limitation by maximizing a surrogate objective while strictly constraining the step size of each update. This ensures that new policies remain within a trust region where the value approximation remains valid and effective.

The optimization objective is formulated as maximizing a surrogate performance function subject to a constraint on the average Kullback-Leibler divergence (KL-divergence) between the old and new policies:

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}(\theta_k, \theta) \quad \text{s.t.} \quad D_{KL}(\pi_\theta || \pi_{\theta_k}) \leq \delta, \tag{2.8}$$

where θ_k represents the parameters from the previous iteration and δ is a hyper-parameter controlling the maximum allowed divergence. The surrogate objective function $\mathcal{L}$ measures how well the updated policy performs relative to the previous one by utilizing the probability ratio between them:

$$\mathcal{L}(\theta_k, \theta) = \mathbb{E}_{s,u\sim\pi_{\theta_k}} \left[\frac{\pi_\theta(u|s)}{\pi_{\theta_k}(u|s)} A^{\pi_{\theta_k}}(s,u) \right]. \tag{2.9}$$

This formulation uses the ratio of probabilities to importance-weight the advantage function, effectively measuring the expected gain over the current policy. The constraint in Equation 2.8 limits the average divergence across visited states to prevent large, destructive policy changes:

$$D_{KL}(\pi_\theta||\pi_{\theta_k}) = \mathbb{E}_{s\sim d^{\pi_{\theta_k}}}[D_{KL}(\pi_\theta(\cdot|s)||\pi_{\theta_k}(\cdot|s))]. \tag{2.10}$$

Here, $d^{\pi_{\theta_k}}$ denotes the state visitation distribution under the old policy. By solving this constrained optimization problem TRPO guarantees monotonic performance improvement during training. The core challenge when attempting to apply TRPO in real-world scenarios is the complexity of components that arises mainly by the usage of second-order methods.

2.1.2. Proximal Policy Optimization

PPO aims to retain the data efficiency and reliable performance of TRPO while simplifying the optimization constraints by utilizing a clipped surrogate objective. Schulman et al. introduced two variants: *PPO-Clip* and *PPO-Penalty*. Since this thesis employs the PPO-Clip variant, the following explanation focuses exclusively on this formulation. PPO-Clip utilizes a clipping mechanism to reformulate the update without requiring additional constraints:

$$\theta_{k+1} = \arg\max_\theta \mathbb{E}_{s,u\sim\pi_{\theta_k}}[L(s,u,\theta_k,\theta)]$$

Typically, multiple update steps are performed using stochastic gradient descent over several epochs of minibatches collected from the current policy [20]. The surrogate loss function L^{CLIP} is defined as:

$$L^{\text{CLIP}}(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a),\quad g(\epsilon, A^{\pi_{\theta_k}}(s,a))\right)$$

where ϵ is a hyperparameter bounding the magnitude of the policy update to ensure stability [20]. The function $g(\epsilon, A)$ is defined as:

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0. \end{cases}$$

Schulman et al. also utilize the addition of entropy as it has been shown to improve exploration by discouraging premature convergence to suboptimal deterministic policies [20–22]. Consequently, the combined loss function to be minimized is updated to:

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{\mathrm{VF}}(\theta) + c_2 S[\pi_\theta](s_t) \right]$$

where $L^{\mathrm{VF}}(\theta)$ denotes the value function loss, $S[\pi_\theta](s)$ denotes the entropy bonus, and c_1 and c_2 are scaling factors. In order to gradually reduce the influence of the entropy bonus during training, the entropy coefficient c_2 is exponentially decayed from an initial value $c_{2,\mathrm{start}}$ towards a minimum value $c_{2,\mathrm{end}}$. The coefficient at training step k is calculated as

$$c_2(k) = \max\left(c_{2,\mathrm{end}}, c_{2,\mathrm{start}} \lambda_{\mathrm{decay}}^k \right),$$

where the decay factor λ_{decay} is chosen such that the minimum value is reached after N training steps:

$$\lambda_{\mathrm{decay}} = \left(\frac{c_{2,\mathrm{end}}}{c_{2,\mathrm{start}}} \right)^{1/N}.$$

2.2. Multi Agent Reinforcement Learning

While single agent RL offers a robust framework for solving a wide range of control problems, many real-world applications are more accurately modeled by using an MARL approach. These include scenarios in the aerospace domain such as the coordination of autonomous Unmanned Aerial Vehicles (UAVs) [23] or conflict avoidance in dynamic thunderstorm environments [24].

MARL problems can be classified acoording to their setting into fully-cooperative, fully competitive or mixed competition and cooperation. While competitive games typically model fully adversarial processes, mixed cooperation and competition settings often emerge in team competitions. Fully-cooperative problems and associated algorithms are of particular interest for a wide range of possible applications.

In a fully-cooperative setting a group of agents aims to maximize a collective reward they receive. They can be modelled using a Decentralized partially observable Markov Decision Process (Dec-POMDP):

$$\langle \mathcal{A}, \mathcal{S}, \mathcal{U}, P, r, \mathcal{Z}, O, n, \gamma \rangle$$

where $\mathcal{A} := \{a_1, ..., a_n\}$ denotes the set of n agents. As in single agent RL, $s \in \mathcal{S}$ describes the true state of the environment at time step t. At each time step each agent a chooses an action $u_t^a \in \mathcal{U}$ forming the joint action $\mathbf{u}_t \in \boldsymbol{\mathcal{U}} \equiv \mathcal{U}^N$ causing a transition according to the state transition function $P(s_{t+1} \mid s, \mathbf{u}) : S \times \mathbf{U} \times S \to [0, 1]$. Agents then receive a shared team reward $r_t = r(s_t, \mathbf{u_t})$. Each agents local action-observation history is denoted by $\tau_t^a \in \mathcal{T}_t := (\mathcal{Z} \times \mathcal{U})^t \times \mathcal{Z}$; the set of all agent's histories by $\boldsymbol{\tau}_t$. Agents choose their action using a decentralized policy $u_t^a \sim \pi^a(\cdot|\tau_t^a)$ that is based only on the individual action-observation history. In the cooperative setting, a team of agents attempt to learn a joint policy $\pi(\boldsymbol{u}|\boldsymbol{\tau}_t) := \prod_{a=1}^{N} \pi^a(u^a|\tau_t^a)$ to maximize the expected discounted reward $J(\pi) \doteq \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$.

MARL is characterized by a unique set of challenges that arise compared to single agent RL. These include:

1. Non-stationarity: Due to the nature of multiple agents simultaneously updating their policies the assumption of a stationary transition distribution is violated making learning in a decentralized manner difficult.
2. Credit assignment problem: Determining how much each agent's individual action contributed to the shared reward poses a major difficulty ([25])

Alongside these core problems there are a wide range of additional challenges associated with MARL problems.

Algorithms for deriving a policy in a multi agent setting can be broadly categorized as follows:

1. Fully centralized (also reffered to as Centralized Training Centralized Execution (CTCE))
2. Independant Learning (IL)
3. Centralized Training Decentralized Execution (CTDE)

Fully centralized approach

A fully centralized approach reformulates the MARL problem as a single-agent RL task. The centralized policy takes all local observations z_t^a as input and directly outputs the joint action $\mathbf{u}_t$. In addition to scaling issues which arise when using this approach on a large number of agents Sunehag et al. also suggests that this approach may lead to sub-optimal policies. In their work they observed central learners converging to a "lazy agent" policy where action is dominated by a single agent while remaining agents are passive [26].

Independant Learning

IL can be considered as fully decentralized as there is no communication between agents during training or inference although some algorithm implementations do make use of parameter sharing to reduce training cost and accelerate convergence. This approach enables easy implementation of MARL algorithms by utilizing existing single agent algorithms while mitigating the scaling issues encountered by fully centralized learning. Because the other agents are considered part of the environment the environment becomes non-stationary which violates the assumption made in

traditional RL systems and thereby may lead to suboptimal policies. Lacking the convergence guarantees has led to this approach being mostly disregarded in research of state of the art algorithms. However, some work that suggests IL may still hold significant potential in real-world applications. Witt et al. show that in some cases IL matches or even outperforms state-of-the-art algorithms utilizing centralized training [27].

Centralized Training Decentralized Execution
CTDE combines the benefits of central learning (full observability and stationary environment) with decentralized execution offering a robust framework for a wide range of real-world applications. This is usually achieved by having individual agents pick actions based on local observations while using information avilable at training time to improve the individual agent policies. CTDE has been the focus of much of the RL research as a result of it's flexibility and generally high performance.

2.2.1. Multi Agent PPO

This section describes Multi Agent PPO (MAPPO) in detail, as it constitutes the core algorithm used in this work. The motivation for selecting MAPPO is discussed in Section 3.3.

MAPPO is a CTDE algorithm based on the single-agent PPO framework. In contrast to other MARL algorithms, such as QMIX [14], which employ specialized architectures to decompose the shared reward signal in order to solve the credit asignment problem, MAPPO directly broadcasts the same reward to all agents [28]. The algorithm employs two separate neural networks: a critic that estimates the value function $V_\phi(s)$ with parameters ϕ, and an actor that represents the policy π_θ with parameters θ.

During training, the critic has access to centralized information, such as the global state s or the combined action-observation $\boldsymbol{\tau}_t$ of all agents. This centralized value estimation allows the critic to account for interactions between agents when evaluating their joint behavior. In contrast, the policy of each agent is conditioned only on its locally available observations. Since the critic is required only during training, it can

be discarded during execution, allowing the learned policies to be deployed in a fully decentralized manner.

2.3. Polars

Throughout the implementation, the Python dataframe library *Polars* [29] is utilized to improve performance. While Polars is similar to the widely used dataframe library *Pandas*, it differs in terms of both its data layout and several key design aspects. Polars uses Apache Arrow as its underlying data architecture while providing its own implementations of many data types. A key advantage of Polars is its use of a Domain Specific Language (DSL) in combination with a query optimizer. Although implementing procedures using Polars expressions typically requires greater development effort, this approach enables efficient parallelization across all available CPU cores. Another important advantage is Polars' support for the list data type, which is used extensively throughout this work to handle the dynamic nature of the task at hand. Lists allow collections of variable-length elements to be stored within individual dataframe cells, providing greater flexibility than fixed-size arrays.

3. System Architecture

This chapter describes the architecture chosen for the overall system. First the high level concept of the proposed system is introduced and important concepts explained. Subsequently the structure of the baseline trajectory, which is the core of the proposed architecture, is defined. Finally, the MARL system design is described.

3.1. High level concept

The high level concept of the overall system is based on the approach introduced by Baneshi, Cerezo-Magaña, and Soler (2026). They propose a 2 stage approach for optimizing trajectories on a network scale. In a first step, individual routes are optimized in regards to their environmental impact by using existing tools. This step is reffered to as the *micro-scale climate-optimal trajectory planning*. In a second step the output of this first step acts as a baseline in the MARL system that balances climate impact and operational managability. The reward is of the structure

$$r^{\text{total}} = s_1 \cdot r^{\text{complexity}} + s_2 \cdot r^{\text{climate}}$$

where s_1 and s_2 are scaling factors. The climate reward r^{climate} is calculated by comparing the climate-ideal trajectory to the baseline trajectory generated in the micro-scale climate-optimal trajectory planning phase. The complexity reward $r^{\text{complexity}}$ is a measure of the airspace complexity at a given time t and is the primary optimization target. The implementation details of the individual reward terms is detailed in section 4.2.4. Agents control individual aircraft following the pre-computed baseline route. At every timestep t all agents pick an action u_t^a to change their speed

and altitude based on an observation z_t^a that comprises individual parameters as well as data on neighboring aircraft. This action space is chosen as research shows that the most influential factor is the altitude and time [31]. Lateral changes would also greatly increase the engineering difficulty as the action space grows exponentially and ensuring agents reach their target proves difficult.
The action space is implemented as a discrete action space as this most closely resembles the structured airspace. Altitude changes are therefore generally constrained to full flight levels (100ft). The specific actions chosen for implementation are described in section 3.3. The decision to model a discrete action space also forces constraints on the baseline trajectory which is described in the following section.

3.2. Baseline Trajectory Structure

This section defines the required structure of the baseline trajectory which is at the core of the proposed architecture.

The baseline trajectory is a set of 4D points (longitude, latitude, altitude, speed) representing the climate-optimal route. As described in the previous section, actions in the environment are picked at a globally aligned time step t instead of individually running action intervals. While reducing complexity and boosting performance this decision introduces a unique problem: As agents can change their own speed v_{real} it is impossible to pre-calculate the baseline points in a way as to align them with the fixed timespan between actions. This issue is illustrated in figure 3.1.

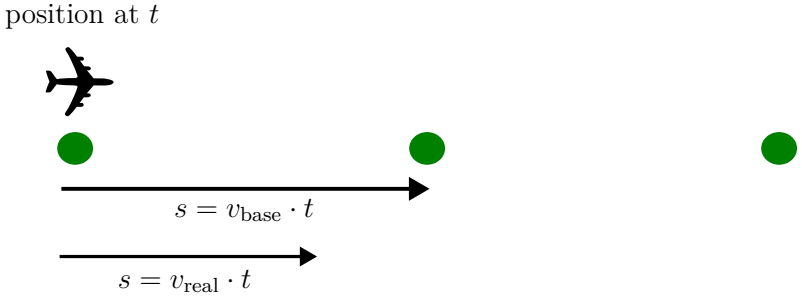


Figure 3.1.: Trajectory with pre-computed decision points. Own illustration; aircraft icon from [32]

One solution to this problem could be achieved by dynamicly iterpolating agent positions. This would, however remove any computational advantage the aligned decision time offers. Therefore a different strategy is adopted. By calculating many additional points along the trajectory, the exact aircraft location at time $t+1$ can be approximated by simply choosing the closest pre-computed point as the new exact position at time $t+1$. This strategy is illustrated in figure 3.2.

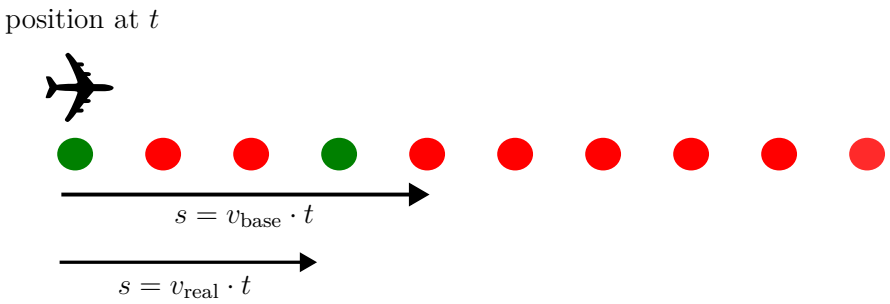


Figure 3.2.: Trajectory with dense baseline points and decision point approximation. Own illustration; aircraft icon from [32]

While approximating the position by simple rounding does introduce some bias, it can be siginificantly lowered by decreasing the distance between points.
For this thesis the distance d between points is set at 5 kilometers; the time Δt between update steps at 2 minutes. These values are chosen specifically to ensure that the approximation stays precise enough and that the discrete transitions described in the previous section can be implemented. It is assumed that aircraft can reasonably achieve a 1 flight level climb in between points regardless of their speed (which is constrained by the performance boundaries of the aircraft). It is also argued that a decision interval of two minutes, while shorter than most present-day ATC operations, more closely resembles realistic operational cadences compared to the ten-second decision frequency used by Baneshi, Cerezo-Magaña, and Soler (2026). While this does increase realism it also significantly impacts the ability of agents to react to changes in the environment.

3.3. MARL system design

This section details the high level design of the MARL system itself. MAPPO was chosen as the algorithm for the system as it has shown considerable performance despite it's relative simplicity compared to other aproaches. The ability to use MAPPO in a CTDE setting is also a major benefit as the decentralized approach offers high robustness and resilience compared to a fully centralized approach.

Policy sharing (i.e., $\theta^1 = \theta^2 = \ldots = \theta^N = \theta$) is employed across actors, as it has been shown to improve performance and accelerate convergence. Furthermore, policy sharing is particularly beneficial in the dynamic environment considered in this work, where the number of active agents can change over time. As agents dynamically enter and leave the simulation, the number of active aircraft varies accordingly. Consequently, the number of agent observations is not fixed, making it infeasible to provide the critic with a simple concatenation of all agent observations. To resolve this issue, the critic is provided with a *global* observation $\omega_t \in \Omega$ that is generated by a seperate observation function O_{global}. The calculation of this observation is detailed in section 4.2.3.
As described previously, all active agents pick a speed $u^a_{v,t}$ and an altitude action $u^a_{h,t}$ at a globally aligned time step t. The available altitude actions are:

- + 3 flight level
- + 2 flight level
- no change
- -2 flight level
- -3 flight level

Analogous to the altitude actions, the speed actions follow the same symmetric, discrete structure. Each agent may choose between two magnitudes of acceleration or deceleration, or maintain its current speed. This symmetry keeps the action space consistent across both control dimensions and simplifies the network architecture, as

both action heads share the same number of discrete outputs. The available speed actions are

- + 30 knots
- + 20 knots
- no change
- -20 knots
- -30 knots

These action increments were chosen such that they can be reasonably achieved between two consecutive decision points, regardless of the actor's current speed. This choice significantly simplifies the action validation logic, as there is no need to explicitly verify climb, descent, or acceleration rates. Instead, it is sufficient to enforce the corresponding absolute performance limits.

4. Implementation

This chapter details the implementation of the system architecture discussed in the previous chapter. It is split in three parts. First, the data loading and preprocessing for the generation of the baseline trajectory is outlined. Second, the implementation of the environment and ascociated logic is described. Finally, the MARL system actor and critic networks as well as the rollout buffer and overall training process are presented.

4.1. Baseline Generation

This section details the processes involved in preparing the baseline trajectories for the training process.

Data generation consists of four phases which will be detailed in the following sections:

1. Loading data
2. Traffic data preprocessing
3. Generating individually optimal routes using PyTOM
4. Altitude and speed adjustments (consisting of trajectory reconfiguration and smoothing)

The process is also visualized in greater detail in figure 4.1.

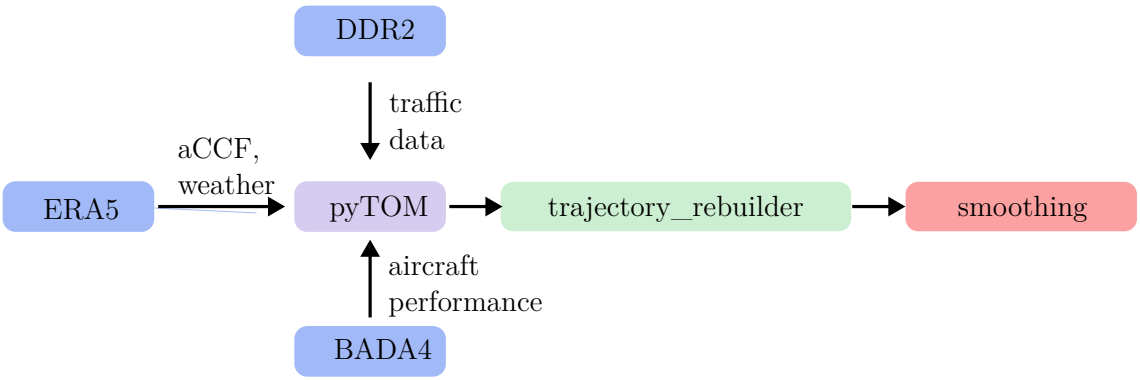


Figure 4.1.: Baseline trajectory generation

4.1.1. Loading Data

There are three primary types of data used as input for the microscale trajectory generation process, which are described in greater detail in the following sections:

1. Air traffic data
2. Aircraft performance data
3. Weather / climate data

The **air traffic data** is sourced from the European Organisation for the Safety of Air Navigation (Eurocontrol)'s Demand Data Repository 2 (DDR2) dataset, which is provided in a proprietary CSV-like format. The dataset contains flight trajectories over continental Europe, with datasets covering a 24-hour period. Each trajectory is represented as a sequence of path segments, with each segment specifying its start and end longitude, latitude, altitude, and time. Additional information, including the aircraft type and the origin and destination airports of each flight, is also provided.

Aircraft performance data is obtained from Eurocontrol's Base of Aircraft Data (BADA) version 4 model and provides the aircraft-specific performance parameters required for the route generation process. This data includes fuel burn rates necessary for the computation of climate-ideal baselines. As BADA 4 does not cover all aircraft types currently in service, unsupported aircraft types are mapped to similar supported

aircraft types where possible. This mapping allows approximately 95% of the available trajectories to be used for training.

Atmospheric weather data is sourced from the European Centre for Medium-Range Weather Forecasts (ECMWF)'s ECMWF Re-Analysis (ERA) 5 reanalysis dataset. The weather data is used to generate algorithmic Climate Change Function (aCCF) datasets, which represent the atmospheric conditions as four-dimensional arrays with longitude, latitude, altitude, and time as dimensions. The aCCF data provides the expected warming potential, expressed in Kelvin, for different emission types per kilogram of fuel burned at a given location and time. A merged variable, $aCCF_{merged}$, is also provided, representing the total expected warming potential across all considered emissions, including contrail formation. The dataset has a spatial resolution of $0.1° \times 0.1°$, a temporal resolution of 1 hour, and a vertical resolution of 2,500 ft.

4.1.2. Traffic data preprocessing

Training data is generated by filtering air traffic data to a square region measuring 500 by 500 nautical miles, with its center randomized within Central Europe. Both the geographic region and the time window are selected to ensure sufficiently high traffic density for the generated scenarios to require a trade-off between climate impact and air traffic complexity. The time window is restricted to 11:00 AM to 5:00 PM, which corresponds to a period of comparatively high traffic activity. This avoids sparse scenarios in which aircraft can follow their baseline trajectories without significant interactions or conflicts.

Randomizing the center of the region provides diverse traffic patterns while preventing the training data from being overly dependent on specific local airspace characteristics.

The calculation of each square region boundaries is achieved by utilizing the python library *pyproj* to ensure precise geospatial calculation. Filtering is performed using a fully parallelized implementation of the Liang–Barsky line clipping algorithm [33] which is detailed below:

The line segment between

$$P_1 = (x_1, y_1), \qquad P_2 = (x_2, y_2)$$

is represented parametrically as

$$\begin{aligned} x(t) &= x_1 + t\Delta x, \\ y(t) &= y_1 + t\Delta y, \end{aligned} \qquad t \in [0, 1],$$

where

$$\Delta x = x_2 - x_1, \qquad \Delta y = y_2 - y_1.$$

For a rectangular clipping window, the four boundary constraints can be written in the standard Liang–Barsky form

$$p_i t \le q_i, \qquad i = 1, 2, 3, 4,$$

with

$$\begin{aligned} (p_1, q_1) &= (-\Delta x,\ x_1 - x_{\text{left}}), \\ (p_2, q_2) &= (\ \ \Delta x,\ x_{\text{right}} - x_1), \\ (p_3, q_3) &= (-\Delta y,\ y_1 - y_{\text{bottom}}), \\ (p_4, q_4) &= (\ \ \Delta y,\ y_{\text{top}} - y_1). \end{aligned}$$

Each constraint produces a bound on the parameter t. The valid interval is therefore

$$t_{\text{enter}} = \max\left(0, \max_{p_i<0} \frac{q_i}{p_i}\right), \qquad t_{\text{exit}} = \min\left(1, \min_{p_i>0} \frac{q_i}{p_i}\right).$$

The segment intersects the clipping window when

$$t_{\text{enter}} \le t_{\text{exit}}.$$

In the parallel implementation, the four (p_i, q_i) pairs are evaluated simultaneously, followed by parallel maximum and minimum reductions to obtain t_{enter} and t_{exit}. By implementing the algorithm using multiple polars expressions, the data required for

the generation of micro-scale climate-optimal trajectories can be filtered quickly and efficiently.

4.1.3. Generating micro-scale climate-optimal trajectories

The most important step of baseline trajectory generation is the optimization of individual flight segments obtained from the previous steps. For this task the Python Trajectory Optimization Module (PyTOM) tool from the German Aerospace Center (DLR) Institute for Air Transport (LV) is utilized. This tool optimizes routes in regard to a weighted sum of the climate impact and economical factors (modelled by kerosine consumption) by applying a graph based search to the problem [34]. PyTOM uses the DLR pyBADA4 library for aircraft performance data and aCCF datasets as its data source. The output trajectory consits of a 4D trajectory (longitude, latitude, altitude and ground speed). The altitude profile generated by PyTOM for an example flight can be seen in figure 4.2. It can be observed that the generated roite maneuvers around areas of high climate sensitivity. The climate sensitivity is shown as a vertical countour from purple (low climate sensitivity) to yellow (high climate sensitivity)

While PyTOM also does optimize trajectories in the lateral dimension, these changes are much less impactful and therefore less common and of limited scale. This can be seen in figure 4.3. The route generated by PyTOM is displayed in red while the shortest path is shown as a black dotted line.

As the optimization of an individual route takes approximately 3–5 minutes, parallelization is necessary to generate the required datasets within a reasonable timeframe. Since PyTOM could not be feasibly adapted to run on GPUs, the Python library *multiprocessing* was used to parallelize the workload across all available CPU cores. To further reduce dataset generation times, multiple nodes of the DLR High Performance Computing (HPC) cluster Computer for Advanced Research in Aerospace (CARA) were used to run the training script in parallel. Despite the applied optimizations and parallelization, the computational effort required to generate the baseline trajectories remains substantial. Therefore, the baseline trajectories are

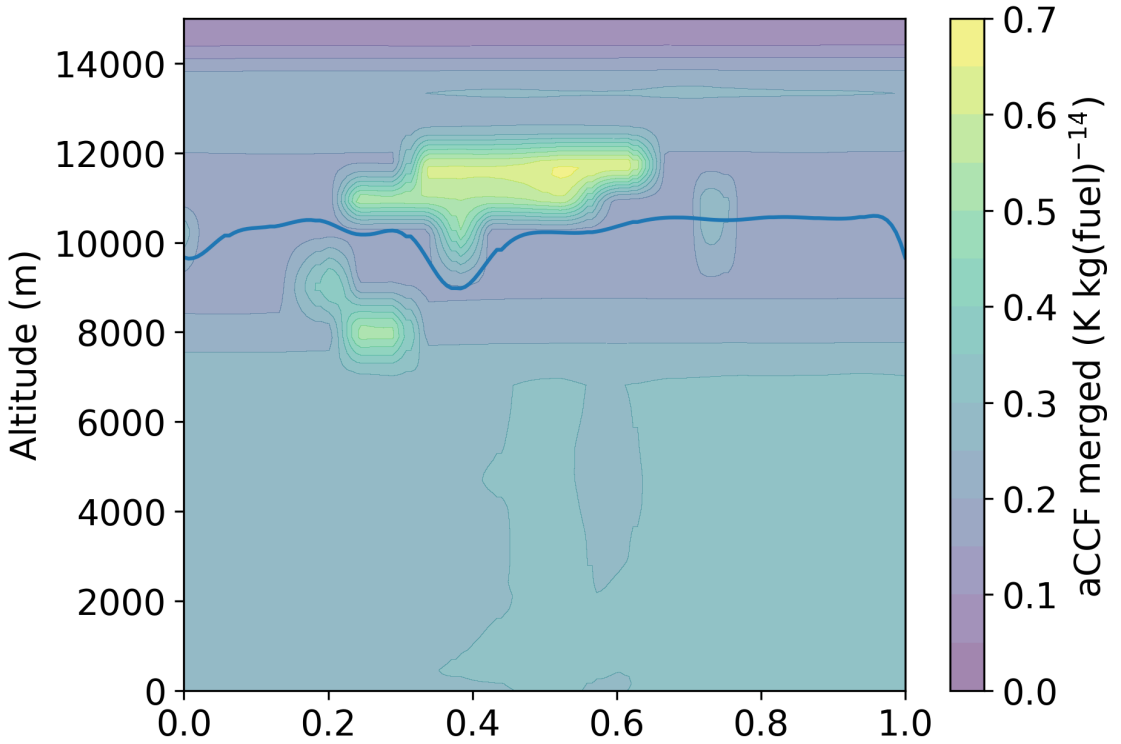


Figure 4.2.: Altitude profile generated by pyTOM

precomputed separately rather than generated online during training, avoiding the considerable computational overhead that would otherwise be incurred during the training process.

4.1.4. Altitude and speed adjustments

As the points generated by PyTOM stem from a graph based optimization, the resulting trajctory does not conform to the structured airspace. To address this issue and transform the raw trajectory into a baseline as described in section 3.2 a multi-step approach is taken. The first step consists of using the *trajectory_rebuilder*, a tool developed at the DLR Institute for Air Transport specifically for adjusting the altitude values produced by PyTOM. The result of this process can be seen in figure 4.4

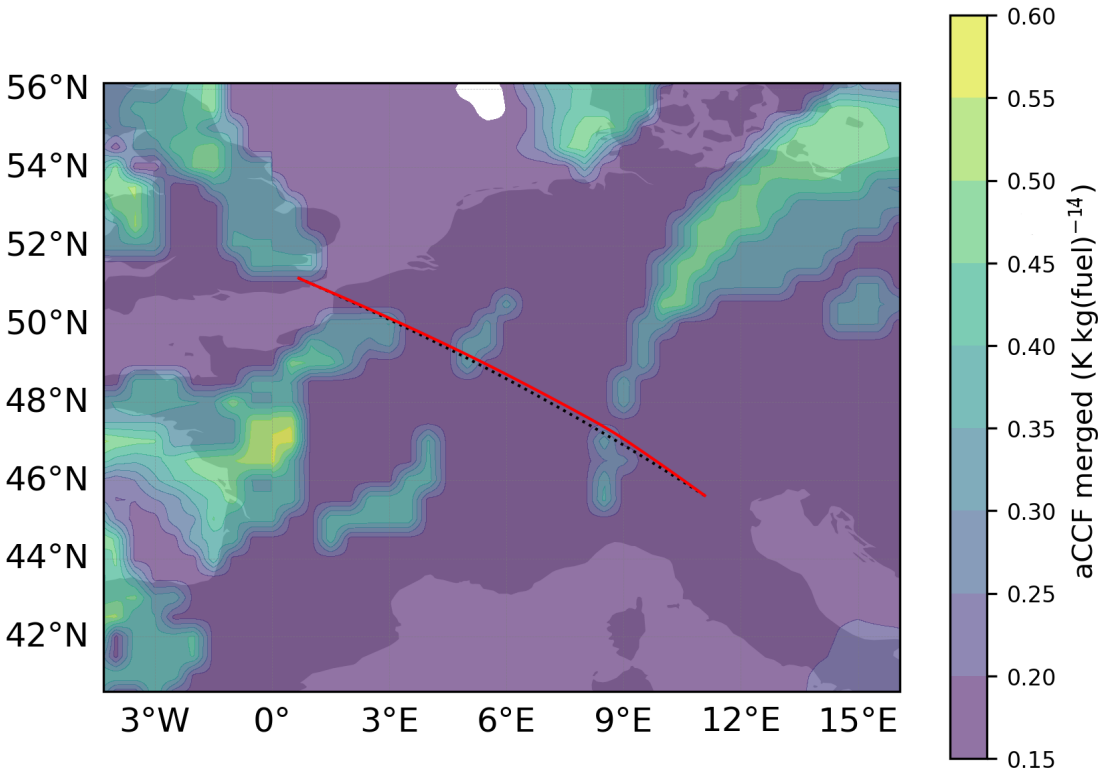


Figure 4.3.: Route generated by pyTOM (red) vs the shortest route (black, dotted)

The red line corresponds to the original altitude profile as generated by pyTOM. It can be observed that the original trajectory consists of many local minima and maxima and is characterized by constant change instead of the climb and cruise phases observed in air traffic operations. The trajectory rebuilder adjusts this trajectory by snapping to minima and maxima and using a user-defined, fixed climb / descent rate. It also takes into account the structured nature of the european airspace. Flights going in the east direction (0 - 179 degree bearing) are typically asigned flight levels starting with odd numbers (e.g. flight level 290, 310, 330) while flights going west (180 - 359 degrees bearing) are asigned flight levels starting with even numbers (e.g. flight level 300, 320, 340).

After rounding the raw altitude values to flight levels, the resulting sequence $\ell = (\ell_1, \ldots, \ell_n)$ may contain transitions exceeding the maximum climb or descent rate permitted between two consecutive points (cf. Section 3.2). To enforce this constraint

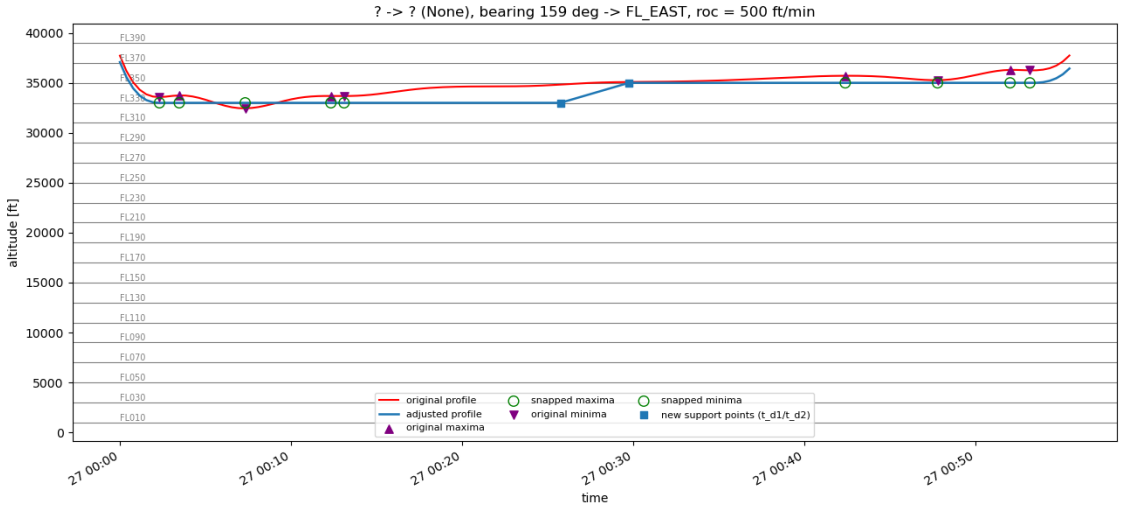


Figure 4.4.: Altitude adjustments made by the trajectory rebuilder

while altering the original profile as little as possible, the largest 1-Lipschitz sequence h not exceeding ℓ at any point is computed, i.e. a sequence satisfying $|h_i - h_{i+1}| \leq 1$ for all i.

This sequence has a closed form [35, 36],

$$h_i = \min_{j \in \{1,\dots,n\}} \Big(\ell_j + |i - j|\Big). \tag{4.1}$$

Intuitively, each index j caps how large h_i can be: since h is 1-Lipschitz, h_i can exceed h_j by at most $|i - j|$, and $h_j \leq \ell_j$. Taking the minimum over all j applies every such cap at once, giving the largest 1-Lipschitz sequence not exceeding ℓ.

A direct evaluation of Equation 4.1 requires $O(n^2)$ operations. Splitting the minimization by the sign of $(i - j)$ yields two one-directional passes, each computable in $O(n)$ via a running minimum:

$$a_i = i + \min_{j \leq i} \Big(\ell_j - j\Big), \tag{4.2}$$

$$b_i = -i + \min_{j \geq i} \Big(\ell_j + j\Big), \tag{4.3}$$

where a propagates the bound left-to-right and b right-to-left. The smoothed sequence

is then simply

$$h_i = \min(a_i, b_i), \tag{4.4}$$

which is equivalent to Equation 4.1 but computable in $O(n)$ instead of $O(n^2)$.

By construction, h never exceeds ℓ and satisfies the maximum climb/descent-rate constraint everywhere. Where the original profile changes too abruptly to satisfy this constraint, h approaches it from below rather than averaging around it, since it is defined as a lower bound rather than a two-sided approximation.

This strategy is also applied to smooth the speed values by scaling them respectively:

$$v_i' = 10 \cdot \mathrm{Lip}_1\left(\mathrm{round}\left(\frac{v}{10}\right)\right)_i$$

where $\mathrm{Lip}_1(\cdot)$ denotes the 1-Lipschitz operator defined by Equation 4.1, applied here in units of 10 kt instead of flight levels. The result of this smoothing applied to the ground speed is shown in Figure 4.6, compared to the original, unsmoothed speed trajectory in Figure 4.5. It can be observed that the adjusted speed is confined to fixed steps of 10 kt, with the rate of change between consecutive points limited accordingly.

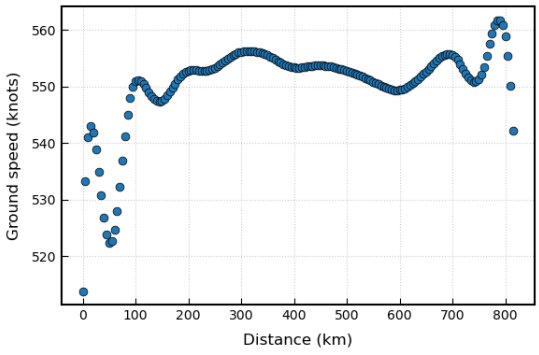


Figure 4.5.: Original speed trajectory

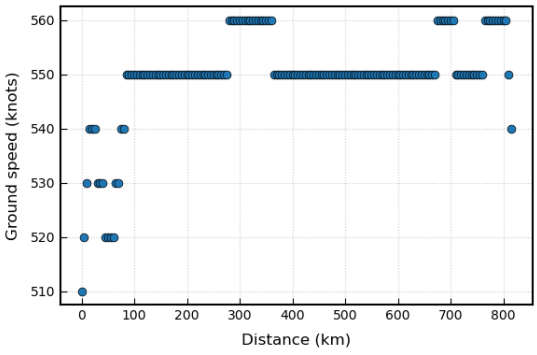


Figure 4.6.: Adjusted speed trajectory

After computing the baseline using the methods presented, it is stored as a parquet dataset for later use in the training.

4.2. Environment

This section details the implementation of the environment for the architecture decribed in chapter 3. The environment provides the functionality required for the successful training of the proposed system. Subsequent sections will detail the different components of the environment:

1. State
2. State Updates
3. Observation Functions
4. Reward Calculation

As MARL requires a lot of state transitions to be able to derive policies it is essential for the environment to be able to perform all required calculations in an efficient manner. Therefore, performance is a key concern in the design and implementation of the individual components.

4.2.1. State

The true state of the environment is represented by the state buffer, which contains all information relevant to the current state. As the core component of the environment, the state buffer requires an efficient data layout. To maximize performance, it is implemented as a dataframe rather than using a more intuitive object-oriented design. Polars is used to manage the dataframe, providing efficient in-memory data storage and fast vectorized operations.

Baseline longitude, latitude, altitude, and ground speed are stored in list columns, while the start time and aircraft type are stored in scalar columns. To keep track of the current position, the indices of the current and previous list elements are stored separately. Since the agent cannot modify lateral movements, this information is sufficient to determine the longitude and latitude positions. The current altitude and ground speed are stored in dedicated columns, as they may differ from their

respective baseline values. All baseline values are imported into the initial state dataframe, as updating them at every simulation step would degrade performance. In addition, a dedicated column is used to indicate whether an aircraft is active or inactive. A row index is also added to serve as a flight ID, enabling individual aircraft to be referenced and other dataframes to be joined efficiently. A full overview of the data layout used for the state buffer can be found in A.3.

4.2.2. State update

Each environment step triggers a state update that is composed of multiple internal operations that are executed in the following order:

1. Time update
2. Update active aircraft
3. Action processing
4. Segment store update
5. Computing observation
6. Complexity reward calculation
7. Climate reward calculation

Time update
The first step consists of updating the internal time to the next interval to prepare for the following steps.

Index update
The second routine in the step function is the index update that is performed for all currently active aircraft. As described in the previous chapter points are generated with a fixed 5 kilometers distance and actions are picked every 2 minutes. Calculation of the next index can therefore be done by basic arithmetic:

$$\text{idx} \leftarrow \text{idx} + \left\lfloor \frac{v \cdot c_{\text{kt_kmh}} \cdot 2}{60 \cdot 5} \right\rfloor \tag{4.5}$$

where idx is the state index and v the current speed in knots and $c_{\text{kt_kmh}}$ a conversion factor from knots to $\frac{km}{h}$.

Update active aircraft

In the third phase aircraft with a new index greater than the length of their baseline trajectory are considered "finished" and are therefore removed from the state dataframe. Aircraft with a start time equivalent to the new current time are also set to active.

Action processing

This step performs both the validation and the application of actions choosen by the agents. Actions are represented by a seperate dataframe containing flight id, altitude and speed changes. They are joined with the state dataframe and validation is performed. The validation consists of comparing aircraft service ceiling and max operating speed with aircraft performance data obtained from the *openAP* python library that provides performance data and other relevant data for air transportation research [37]. If actions submitted by agents are in violation of the performance boundaries; they are replaced with a *no operation* that does not change altitude or speed. After validating the actions, they are applied to the state dataframe.

Segment store update

The segment store is composed of lists used to store the altitude and speed profiles of the last flown segment. They are used to calculate the climate reward by comparison to the baseline. The intracacies of the reward calculation itself are described in-depth in section 4.2.4. After the actions are applied to the state dataframe, the segment store is updated by fetching altitudes and speeds between the previous and current index. Using the state dataframe and the verified actions, the segments are modified to reflect the actions chosen. Speed changes are applied along the entire segment as the maximum speed change of 30 knots was specifically chosen to be realistic within the 5km bounds. Altitude change constraints of at most one flight level per point change are implemented by adding a numpy range.

4.2.3. Observation

The observation z_t^a the agents receive are made up of own aircraft parameters and neighboring aircraft. The own aircraft parameters used in the observation are:

- next altitude
- next speed
- current altitude
- current speed
- altitude difference from the baseline at current time
- speed difference from the baseline at current time
- heading

The next altitude and next speed are computed by a value lookup using the pre-computed next index using equation 4.5. Note that the value obtained through this method has some bias attached as the agent may adjust it's own speed and therefore its next decision point.
The data collected for every neighboring aircraft consists of the following parameters:

- relative altitude
- relative heading
- relative speed
- distance to observer

Neighbor observations are created by aggregating aircrafts in a fixed bounding box and then either 0-padding or sampling to receive a fixed-size observation of 10 entries. This approach was chosen over a N-nearest neighbor observation as it is more performant due to its reliance on only simple comparisons against the coordinates of the bounding box corners. For this thesis an observation bounding box with a side length of 80 km was used.
Observations are transformed from a polars dataframe with array values into two

seperate observations, stored in pytorch tensors. The observation of own parameters for an agent a and time t is denoted as $z^a_{\text{own},t}$; the observation of neighboring aircraft as $z^a_{\text{neighbor},t}$.
The global observation $\omega_t \in \Omega$ used as input for the critic, is generated by concatenating individual agent parameters

- altitude
- speed
- longitude
- latitude
- heading
- altitude difference from the baseline
- speed difference from the baseline

Zero-padding / sampling is used to generate a fixed size matrix that holds 400 entries.

4.2.4. Reward calculation

Calculation of the reward is an integral part of the environment and makes up the majority of compute time used in each iteration. The reward r_t at timestep t is defined as

$$r_t = c_{\text{complexity}} \cdot r_{\text{complexity, t}} + c_{\text{climate}} \cdot r_{\text{climate},t}$$

where $r_{\text{complexity, t}}$ is the complexity reward at t and $r_{\text{climate},t}$ the climate reward. $c_{\text{complexity}}$ and c_{climate} are scaling factors for complexity and climate reward respectively. In the scope of this work a distinction is also made between training reward and evaluation reward. The training reward includes additional scaling logic that attempts to steer learning in the desired direction. For evaluation in section 5 the *raw* reward

is used to measure real improvements in air traffic complexity and climate impact. Thus, the training reward is expressed as

$$r_t = c_{\text{complexity}} \cdot s_{\text{complexity}}(r_{\text{complexity, t}}) + c_{\text{climate}} \cdot s_{\text{climate}}(r_{\text{climate},t})$$

where $s_{\text{complexity}}(x)$ is the scaling function for the complexity reward and $s_{\text{climate}}(x)$ the scaling function for the climate reward respectively. An essential component of both scaling functions is a running normalization. The normalized value $norm(x)$ is calculated by

$$norm(x) = \frac{x - \mu}{\sigma + \varepsilon}$$

where ε is a smoothing constant for numerical stability, and $\sigma = \sqrt{v}$ is the running standard deviation. The running mean μ and running variance v are updated recursively at each step t using the momentum parameter m:

$$\mu_t = m \cdot \mu_{t-1} + (1 - m) \cdot \mu_{\text{batch}}$$
$$v_t = m \cdot v_{t-1} + (1 - m) \cdot v_{\text{batch}}$$

While using the same momentum and epislon, different normalizers are used for complexity and climate rewards.

Complexity reward

The calculation of the complexity reward is motivated by the 2006 *Complexity Metrics for ANSP Benchmarking Analysis* by Eurocontrol [38]. The report defines the *adjusted density* as an proxy for the complexity of an airspace. It is computed by dividing the hours of interactions by the total number of flight hours. An interaction is defined as the simultaneous presence of two aircraft in the same fixed-size cell viewed from each aircrafts' perspective. Cell dimensions used in the study are 20 nautical miles in x and y dimension and a vertical height of 3000 ft.

In contrast to the report, the complexity in the scope of this work is defined as the number of total interaction n at a specific time t. This reduces the computational workload significantly as no cell transitions have to be calculated.

The number of total interactions n is computed by the summation of vertical interactions n_{vertical}, horizontal interactions $n_{horizontal}$ and speed interactions n_{speed}.

The report considers two aircraft to interact vertically if they are simultaneously present in the same cell and have different attitudes (climbing-cruising-descending). As the systam architecture described in section 3 considers aircraft in level flight at each decision time t, the last action u^i_{t-1} is used as a proxy for the attitude instead. Two aircraft are considered to interact hoizontally if the difference between their headings exceeds 20 degrees. Finally, speed interactions are defined as the difference in speeds for a pair of aircraft exceeding 35 knots [38].

For training the scaling function $s_{\text{complexity}}$ is defined as

$$s_{\text{complexity}}(r_{\text{complexity}}) = -norm(\frac{r_{\text{complexity}}}{n^{\beta}_{\text{active}}})$$

where n_{active} is the number of active agents at time t and $\beta = 1.35$ is fixed scaling parameter. The scaling factor β and the nonlinear scaling overall are applied as to accurately account for any bias caused by the number of aircraft in the environment. A detailed data analysis and further explanations as to the calculation of β can be found in the supplemental material A.1.

Climate reward

For calculating the climate reward two different approaches were implemented:

- precise
- fast

The **precise** method calculates the actual climate impact for each segment comparing baseline and actual trajectory after modifications. This is achieved by utilizing the python library *pycontrails* [39] to estime the amount of fuel burned when traversing the segment. The calculation is done using the Poll-Schuman Model developed by Poll et al [40] to provide a simple yet accurate model for aircraft fuel burn and performance. The fuel mass calculated by the model is used in a second step in order to calculate the actual climate impact by performing a multiplication with a scaling factor combining both and non-CO2 effects obtained from the aCCF files described in chapter 4.1.1. Finally, the climate reward $r^a_{\text{climate},a}$ for an agent a at time step t is calculated as

$$r^a_{\text{climate},t} = \frac{c^a_{\text{actual},t-1} - c^a_{\text{baseline},t-1}}{c^a_{\text{baseline},t-1}}$$

where $c^a_{\text{actual},t-1}$ is the climate cost of the trajectory segment flown by the agent as a result of their actions at $t-1$ and $c^a_{\text{baseline},t-1}$ is the climate cost of the baseline trajectory segment

The estimation of the fuel usage is computationally expensive therefore significantly slowing down the overall processing speed of the environment to around $0.04\frac{s}{aircraft \cdot step}$.

Alongside the precise model, a **fast** variant was implemented by utilizing the Mean Average Error (MAE) between baseline and actual trajectory:

$$r^i_{\text{climate},t} = \frac{1}{\text{cur_idx} - \text{prev_idx}} \cdot \sum_{k=\text{prev_idx}}^{\text{cur_idx}} \left(\left| v^i_{\text{actual},k} - v^i_{\text{baseline},k} \right| + \left| h^i_{\text{actual},k} - h^i_{\text{baseline},k} \right| \right)$$

This reward can be computed efficiently due to its fully vectorized implementation in Polars. In contrast, the precise reward computation cannot be parallelized, as the underlying library lacks native support for concurrent execution. Furthermore, the overhead associated with multiprocessing would exceed any potential performance gains, rendering parallel approaches impractical. The fast climate reward can be computed in $0.0005\frac{s}{aircraft \cdot step}$ speeding up overall environment execution by on order of magnitude.

After calculating individual climate rewards, the combined reward is then calculated by computing the mean climate reward per aircraft.

The training scaling function $s_{\text{climate}}(r_{\text{climate}})$ is defined as

$$s_{\text{climate}}(r_{\text{climate}}) = -norm(r_{\text{climate}})$$

4.3. MARL system

This section describes the structure of the actor and critic networks as well as the buffer and the overall training logic. Both networks are implemented using pytorch and common implementation practices are adopted. Network parameters are initialized using Orthogonal initilization ([41]) with a bias of $\sqrt{2}$ for linear layers and 0.01 for output heads.

4.3.1. Actor network

The Actor network consists of two separate input branches that independently process the agent's own state and information about its neighbors. Each branch consists of two fully connected layers with 128 neurons and Rectified Linear Unit (ReLU) activations. Since the number of neighboring aircraft varies across time steps, a binary mask $m \in \{0, 1\}^{10}$ is applied via elementwise multiplication to the neighbor feature vectors, zeroing out the padded (non-existent) neighbor slots before they are processed by the fully connected layers. The outputs of the two branches are concatenated and passed through a linear fusion layer. Finally, two separate output heads generate the action distributions for speed and altitude. The overall architecture is shown in figure 4.7.

4.3.2. Critic network

The Critic network first projects the features of each entity into a 128-dimensional representation. These representations are then processed by three residual blocks, each consisting of two linear layers with a ReLU activation, a skip connection, and LayerNorm. The resulting entity representations are aggregated using masked mean pooling. Finally, the pooled representation is passed through a two-layer MLP to produce a scalar Value $V(s)$. The overall architecture is shown in 4.8.

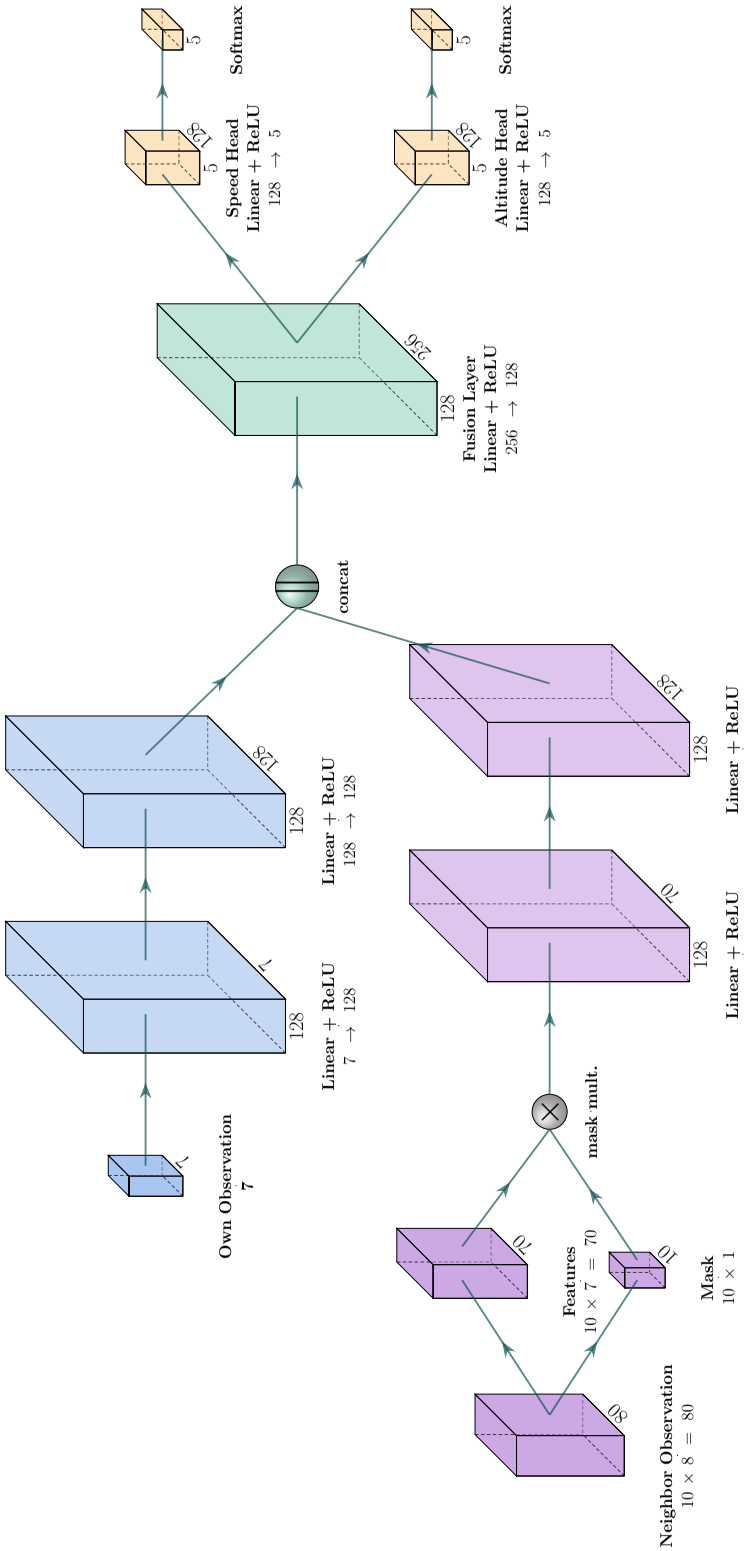


Figure 4.7.: Actor Network. Visualization using [42]

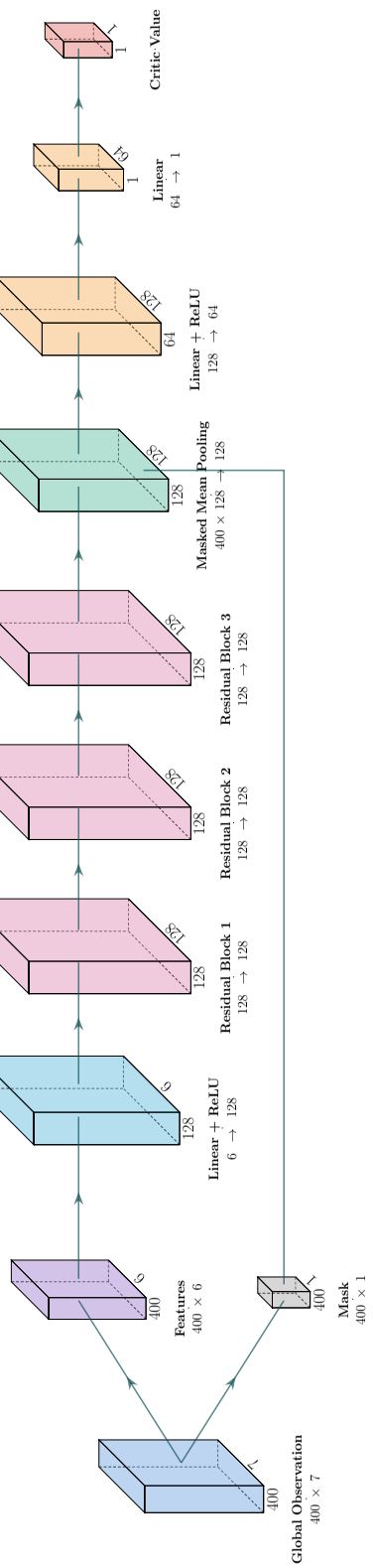


Figure 4.8.: Critic Network. Visualization using [42]

4.3.3. Rollout Buffer

The Rollout Buffer collects action-observation histories $\boldsymbol{\tau}_t$ as well as the global observations ω_t, value estimations v_t and the team reward r_t. For each step the advantage $\hat{A}_t$ is computed by the buffer using Generalized Advantage Estimation (GAE) ([43]):

$$\delta_t = r_t + \gamma\, V(s_{t+1})\,(1 - d_t) - V(s_t) \tag{4.6}$$

$$\hat{A}_t = \delta_t + \gamma\lambda\,(1 - d_{t+1})\,\hat{A}_{t+1} \tag{4.7}$$

where $\gamma \in (0, 1]$ is the discount factor and $\lambda \in [0, 1]$ is the GAE smoothing parameter that trades off bias and variance in the advantage estimate. The flag $d_t \in \{0, 1\}$ indicates episode termination at step t; since termination is defined at the episode level rather than per agent, d_t is shared identically across all agents active at that step. Analogously, r_t denotes the team reward, which is likewise shared across all active agents. As the agents do not have access to the true environment state s_t, the critic is instead conditioned on the global observation ω_t, so that $V(s_t)$ is in practice evaluated as $V(\omega_t)$.

4.3.4. Training Loop

The training loop of the overall system is presented in Algorithm 1. At each training iteration, an environment e is sampled uniformly at random from the pool $\mathcal{E} = \{e_1, \ldots, e_K\}$ and reset, after which a full trajectory is collected by rolling out the current shared policy π_θ until the episode terminates. During this rollout, every active agent $a \in \mathcal{A}_t$ selects an action according to $\pi_\theta(\cdot \mid z_t^a)$, and the resulting transitions, including the global observation ω_t required by the critic, are stored in a rollout buffer $\mathcal{B}$. Once the episode has terminated, GAE is applied to $\mathcal{B}$ to obtain advantage estimates $\hat{A}$ and return targets $\hat{R}$, which are subsequently used to update π_θ and V_ϕ by minimizing the combined loss $L_t^{\text{CLIP+VF+S}}$ introduced in Section 2.1.2 over multiple epochs of minibatches sampled from $\mathcal{B}$.

Since each agent selects a factored action $(u^a_{v,t}, u^a_{h,t})$ consisting of a speed and an altitude component (Section 3.3), the probability ratio used in the clipped surrogate objective L^{CLIP} is computed as the product of the two per-head ratios,

$$\rho^a_t(\theta) = \frac{\pi_\theta(u^a_{v,t} \mid z^a_t)}{\pi_{\theta_k}(u^a_{v,t} \mid z^a_t)} \cdot \frac{\pi_\theta(u^a_{h,t} \mid z^a_t)}{\pi_{\theta_k}(u^a_{h,t} \mid z^a_t)}.$$

A further deviation from a standard MAPPO setting arises from the dynamic size of the agent population: since aircraft enter and exit the airspace throughout an episode, the set of active agents $\mathcal{A}_t$ varies at every timestep t. Consequently, the rollout buffer stores a variable number of per-agent transitions at each step rather than a fixed-size batch, and — as motivated in Section 3.3 — the critic is conditioned on the global observation $\omega_t \in \Omega$ rather than a concatenation of individual agent observations, since the latter would require a fixed and known number of agents.

As PPO is an on-policy algorithm, the rollout buffer $\mathcal{B}$ is discarded after each update step: transitions are used for at most K epochs of minibatch updates before being replaced by data collected under the newly updated policy π_θ in the following training iteration.

Algorithm 1: MAPPO training loop

Input: Environment pool $\mathcal{E} = \{e_1, \ldots, e_K\}$, shared actor π_θ, centralized critic V_ϕ

Output: Trained actor π_θ

for $i \leftarrow 1$ **to** N_{steps} **do**

  `// Collect trajectory`

  $e \sim \mathcal{E}$; `// pick random environment`

  Reset e; $t \leftarrow 0$; $\mathcal{B} \leftarrow \emptyset$;

  while *episode not done* **do**

    foreach *active agent* $a \in \mathcal{A}_t$ **do**

      $(u^a_{v,t}, u^a_{h,t}) \sim \pi_\theta(\cdot \mid z^a_t)$;

    end

    $\mathbf{u}_t \leftarrow \{(u^a_{v,t}, u^a_{h,t})\}_{a \in \mathcal{A}_t}$;

    Execute $\mathbf{u}_t$ in e; observe r_t, $\{z^a_{t+1}\}_a$, ω_{t+1}, done;

    Add $(z^a_t, \omega_t, u^a_{v,t}, u^a_{h,t}, r_t)$ to $\mathcal{B}$ for each active a;

    $t \leftarrow t + 1$;

  end

  `// Advantage estimation`

  $(\hat{A}, \hat{R}) \leftarrow \mathrm{GAE}(\mathcal{B})$; `// via` $V_\phi(\omega_t)$

  `// PPO update`

  foreach *minibatch* $b \subset \mathcal{B}$ **do**

    Compute policy ratio ρ_b and clipped surrogate loss L_{clip};

    Compute value loss L_V using $V_\phi(\omega_b)$;

    $L \leftarrow L_{\text{clip}} + c_V L_V$;

    $\theta, \phi \leftarrow \text{Adam-update}(\nabla_{\theta,\phi} L)$;

  end

end

return θ;

5. Results and discussion

This chapter presents the early results of testing the MARL system. The primary aim of this section is not to present final or conclusive results, but rather to evaluate the proposed system as an integrated whole and to identify potential problems and corresponding solutions. The results are divided into two testing campaigns. For each campaign, the parameters were tuned empirically, and multiple runs were performed. The analysis focuses on the runs that provide the most insightful results. Due to the different parameter configurations used across runs, the results are not directly comparable unless explicitly stated otherwise. Furthermore, due to computational constraints, all results discussed in this chapter were obtained using the *fast* reward configuration.
The remainder of this chapter is structured as follows. First, the policies used as baselines for comparison with the MARL system are designed, and the overall evaluation strategy is introduced. Subsequently, the results of the first testing campaign are presented and analyzed in detail. Based on this analysis, necessary changes to the overall system are identified. The second testing campaign, informed by the findings of the first campaign, is then presented and its results are discussed.

5.1. Evaluation Policies

As the baseline trajectory used during training may contain actions at every point, its effective action frequency can be higher than once every two minutes. Therefore, 2 deterministic policies are proposed as a reference frame. The *no operation (NOOP) Policy*, denoted as π_{NOOP}, applies no changes to speed or altitude, thereby retaining the original speed and altitude. The second policy, referred to as the *Next-Point*

Greedy Policy (NPGP) and denoted by π_{NPGP}, selects the speed and altitude changes that most closely match those required to reach the next point. The policy selects the available discrete changes according to

$$a_v^* = \arg\min_{a_v \in \mathcal{A}_v} |a_v - \Delta v| \,, \qquad a_h^* = \arg\min_{a_h \in \mathcal{A}_h} |a_h - \Delta h| \,. \tag{5.1}$$

where Δv and Δh are the difference between current and next speed and altitude. It should be noted that, despite attempting to match the baseline trajectory, the NPGP does not necessarily achieve the lowest climate reward. This is because the policy can only apply discrete actions at the beginning of each segment, whereas the baseline trajectory may apply changes later within the segment. Consequently, an action that best matches the required change at the end of the current segment (i.e. the next decision point) may result in a high penalty due to an early divergence from the baseline. In contrast, maintaining the current state may result in a lower penalty if the baseline applies an opposing change in the following segment. It should also be noted that the complexity generated by the NOOP policy is naturally lower than that of the NPGP policy, as the NOOP policy does not change altitudes and therefore effectively reduces the vertical interactions, as defined in Section 4.2.4, to zero.

5.2. Initial results

Initial experiments are performed using a policy trained on a random, fixed set of 10 spatial environments. An analysis of the training data used for this campaign can be found in Appendix A.2. A full list of parameters used for the featured training run are provided in the supplemental material A.3. While a single run is selected for analysis, the overall patterns are mirrored across different seeds.
During training, the policy is evaluated every 10 steps on a seperate fixed environment to measure progress. The progress of training is shown in figure 5.1 which shows the cumulative climate penalty (left) and the cumulative complexity of the airspace (right).

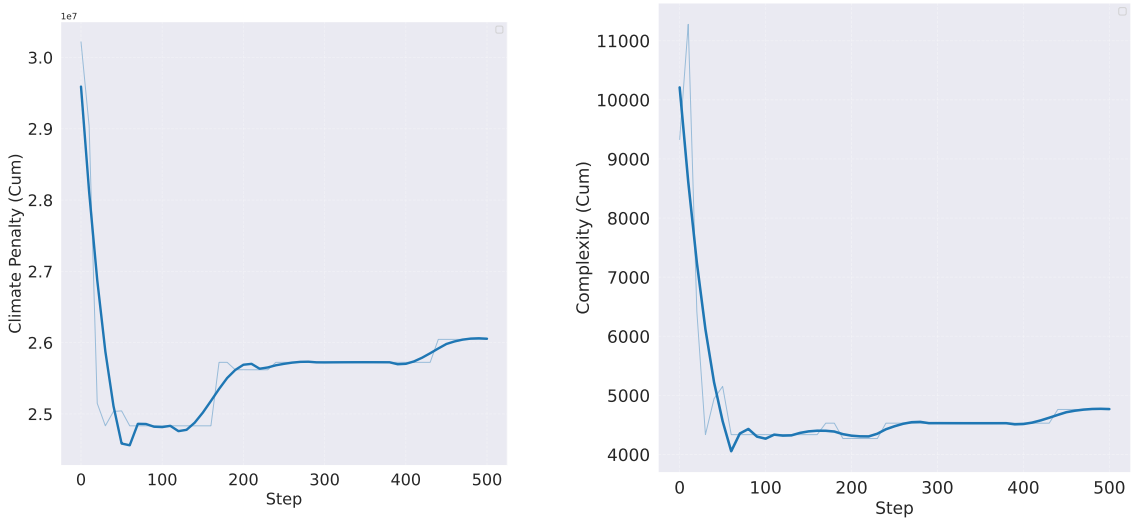


Figure 5.1.: Cumulative evaluation climate (left) and complexity (right) rewards throughout training

Both the climate penalty and airspace complexity decrease substantially during the first 100 policy update steps, indicating rapid initial learning. Around 200 update steps, however, the climate penalty increases sharply before reaching a comparatively stable plateau. This increase coincides with a temporary increase in entropy which is shown in figure 5.2.

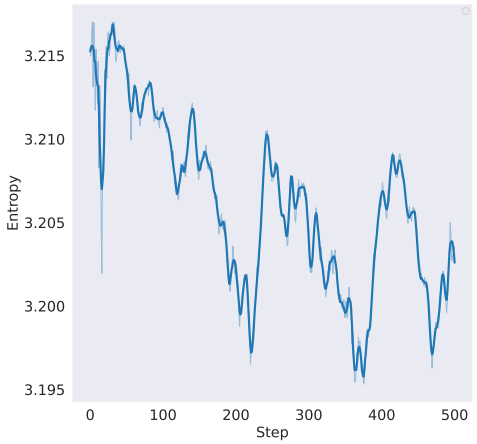


Figure 5.2.: Entropy throughout training

The learning process is particularly notable given the high variance of the underlying reward signal. Figure 5.3 shows an example of the raw climate and complexity rewards observed during training.

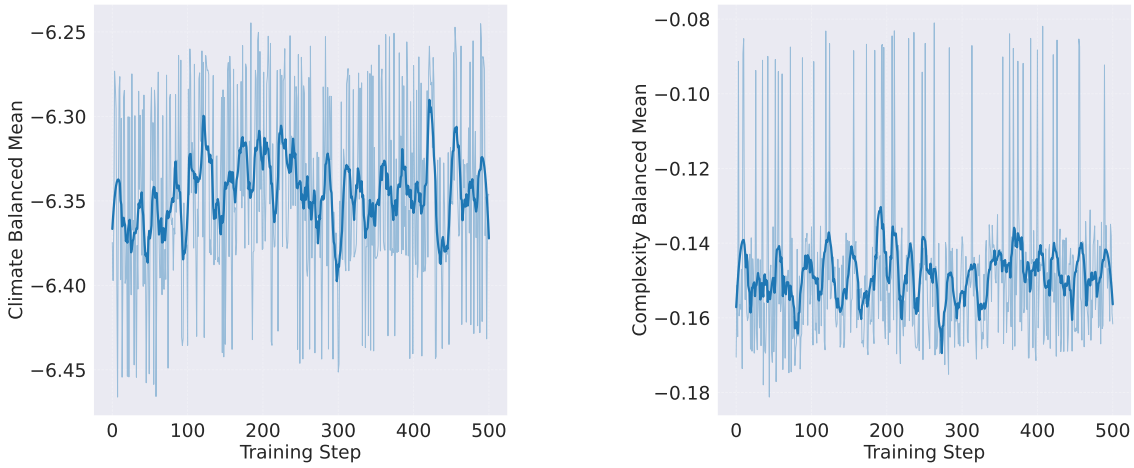


Figure 5.3.: Mean Balanced Complexity (left) and Climate (right) Training Rewards throughout training

Despite substantial fluctuations between individual evaluations, the cumulative evaluation metrics in Figure 5.3 exhibit clear changes during training. This suggests that the policy is able to extract a consistent learning signal from a highly variable reward.

The resulting policy, denoted as $\pi_{\text{RL_initial}}$, is subsequently evaluated against the baseline policies described in Section 5.1. Evaluation is performed on four spatial environments that are not included in the training set, thereby providing an initial assessment of the policy's ability to generalize to previously unseen environments. The results are summarized in table 5.1.

Table 5.1.: Mean, cumulative, minimum, and maximum evaluation metrics across policies

Metric	Statistic	*RL_initial*	NOOP	NPGP
Complexity	Mean	**17.486**	20.81 (-15.98%)	36.53 (-52.13%)
	Cum	**12,520**	14,90 (-15.98%)	26,15 (-52.13%)
	Min	**0**	0	0
	Max	**68**	88 (-22.73%)	132 (-48.48%)
Climate	Mean	$\mathbf{1.115 \times 10^5}$	1.18×10^5 (-5.62%)	1.17×10^5 (-5.02%)
	Cum	$\mathbf{7.984 \times 10^7}$	8.46×10^7 (-5.62%)	8.41×10^7 (-5.02%)
	Min	$\mathbf{3.354 \times 10^3}$	2.66×10^3 (+26.04%)	2.80×10^3 (+19.72%)
	Max	$\mathbf{1.921 \times 10^5}$	2.02×10^5 (-5.06%)	2.02×10^5 (-4.98%)

$\pi_{\text{RL_initial}}$ achieves the lowest mean and cumulative complexity among the evaluated policies. Compared to the two baseline policies, the mean complexity is reduced by approximately 16% relative to NOOP and 52% relative to NPGP. A smaller improvement is observed for the climate metric, where the mean climate penalty is reduced by approximately 5% relative to both baselines. Overall, these results indicate that the initial training setup is capable of producing a policy that improves upon the considered baseline policies on previously unseen environments.

However, the magnitude and speed of the observed improvement warrant a closer examination of the learned behavior. In particular, the policy achieves most of its improvement within the first 100 policy update steps, despite being trained on a highly variable reward signal. This rapid convergence raises the question of which behavioral changes are responsible for the observed improvement. An analysis of the agent's parameters reveals a strong relationship between the agent's speed and the observed reward.

To investigate this relationship in greater detail, a separate training run using a single fixed environment is analyzed. The reduced environmental variability makes the relationship between speed and the individual reward components more apparent. Figure 5.4 (left) shows the agent's mean speed together with the climate penalty throughout training, while Figure 5.4 (right) shows the corresponding development of the airspace complexity.

A clear relationship between agent speed and the two reward components can be

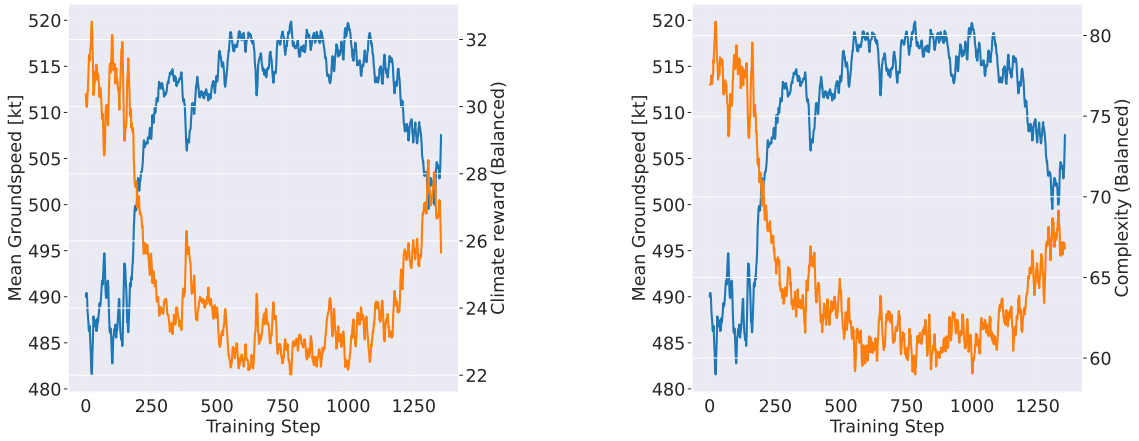


Figure 5.4.: Correlation between Mean Agent Speed (orange) and Complexity (left) Climate Reward (right) during training with a fixed environment

observed throughout training. Increases in speed coincide with reductions in both the climate penalty and airspace complexity. Conversely, reductions in speed are accompanied by increases in the corresponding penalties.
The observed relationship between speed and the reward components suggests that speed may provide a mechanism for reducing the measured penalties. To investigate whether this relationship is reflected in the behavior learned by the policy, the actions selected by the agent are analyzed directly.

Figure 5.5 shows the mean probability assigned to each available action throughout training. Only minor changes in the action probabilities can be observed over the course of training. In particular, the probability assigned to speed-increasing actions increases slightly, while the overall distribution remains broadly similar.

In contrast, the distribution of actions actually selected by the agents exhibits a substantially different behavior. Figure 5.6 shows the mean action selected throughout the evaluation runs during training. The mean action converges to a single action, indicating that the agents consistently select the same action with few exceptions. It can also be observed that this selected action changes from adding 20 knots to adding 30 knots between steps 190 and 240, coinciding with the increase in climate cost and policy entropy observed previously (Figures 5.1 (left) and 5.2).

The observed relationship between speed and the reward components, together

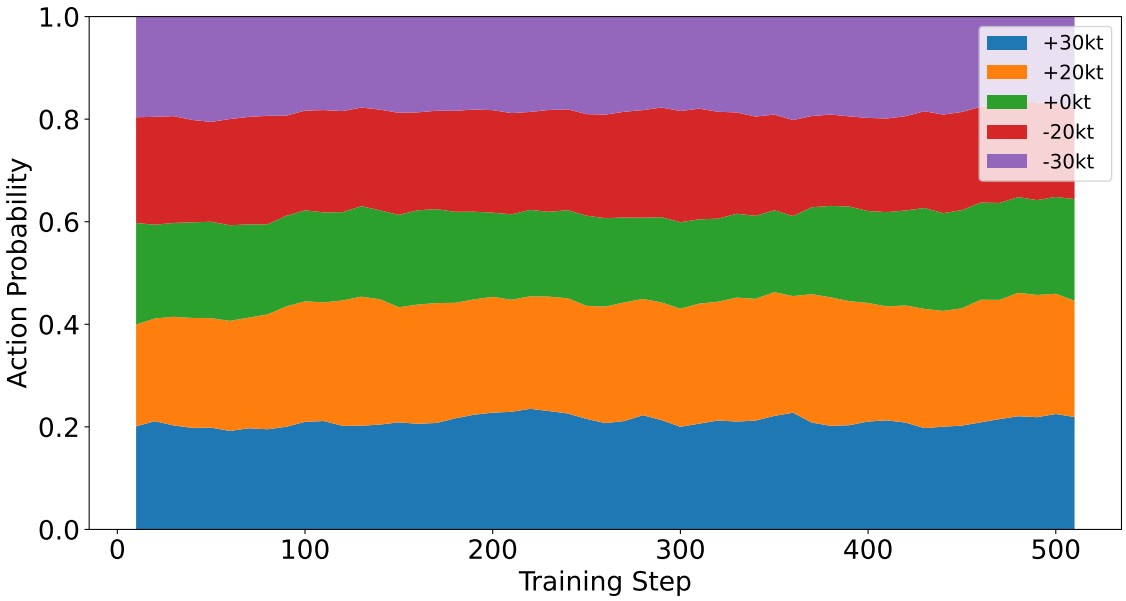


Figure 5.5.: Evaluation mean action probability throughout training

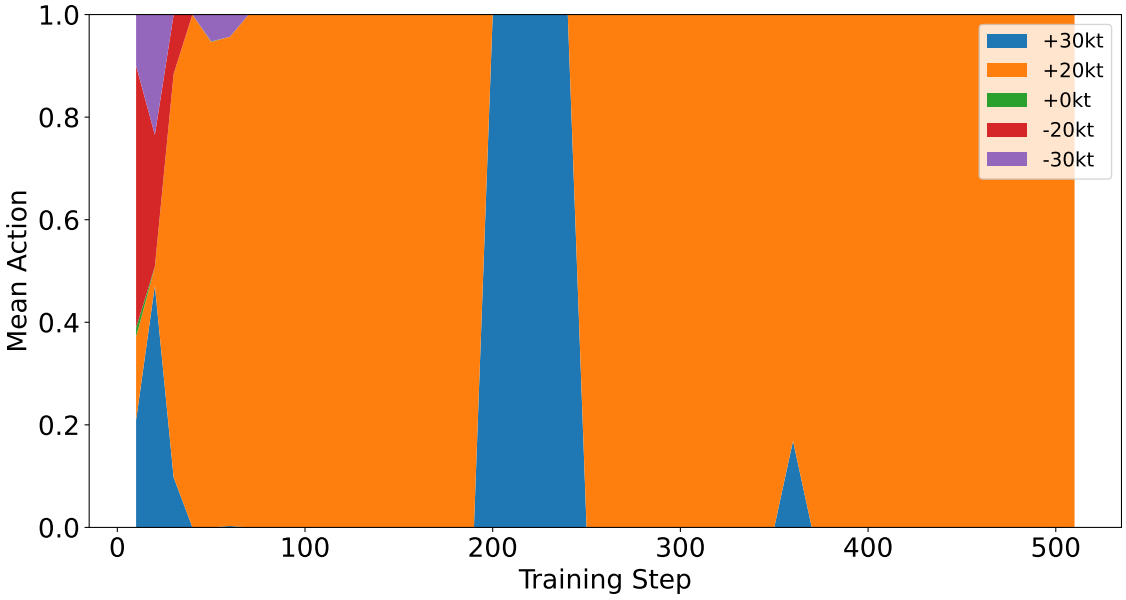


Figure 5.6.: Evaluation mean action throughout training

with the analysis of the selected actions, indicates that the learned policy exploits an unintended property of the reward function. Although only a small change is observed in the action probabilities, the actions actually selected by the agents converge almost entirely to a single speed-increasing action. Increasing the agent's speed therefore provides a simple mechanism for reducing the measured penalties, without necessarily corresponding to an improvement in the intended behavior. This behavior constitutes reward hacking and indicates that the initial reward formulation does not sufficiently constrain the agent's speed.
Nevertheless, the initial experiments demonstrate that the MAPPO training setup is capable of learning a consistent policy and improving the measured evaluation metrics. The results therefore suggest that the primary limitation lies in the reward formulation rather than the underlying training procedure.

5.3. Speed Penalty

To address the issue presented in the previous section, the following experiments introduce an explicit speed penalty to discourage the observed behavior. Instead of being applied at each time step t, the penalty is assigned upon completion of an aircraft's trajectory, allowing greater flexibility in the learned policy.

5.3.1. Penalty Implementation

Rather than comparing the actual speeds, the policy is evaluated by comparing the number of steps taken by an agent with the number of steps required by the corresponding baseline trajectory to reach the final state. In this way, speed is encoded implicitly through the relative number of steps. Bootstrapping is performed separately for each environment to determine the number of steps required by baseline b, denoted by N^b_{base}:

$$N^b = \min\left\{t : i^b_t \geq L^b\right\} \tag{5.2}$$

where L^b is the list length (i.e. the amount of base points) of baseline b and i_t^b is the index of baseline b at time t which is updated by the same recursive equation 4.5 as the current index in the environment:

$$i_{t+1}^b = i_t^b + \left\lfloor \frac{v^b(i_t^b) \cdot c_{\text{kt_kmh}} \cdot 2}{60 \cdot 5} \right\rfloor, \qquad i_0^b = 0 \tag{5.3}$$

.

The penalty $r_{\text{fin, t}}$ at step t is then calculated by

$$r_{\text{fin, t}} = -\frac{C}{N_t} \sum_{i=1}^{N_t} \begin{cases} (s_{b,i} - s_{r,i})\, P_E, & s_{b,i} \geq s_{r,i}, \\ |s_{b,i} - s_{r,i}|\, P_L, & s_{b,i} < s_{r,i} \end{cases}$$

where N_t is the number of agents that finished their trajectory in step t. $s_{b,i}$ is the number of steps of the baseline for agent i and $s_{r,i}$ is the number of steps actually taken by the agent i. P_E and P_L are two different scaling factors for early and late finish. Finally, C is a constant that represents the mean complexity per agent and step. It was calculated based on a set of 10 trajectories collected using the π_{NPGP} policy.

For all experiments, the following parameter values were used:

Parameter	**Value**
C	0.180964
P_E	0.8
P_L	0.1

It should be noted that P_E is significantly larger than P_L, resulting in a substantially higher penalty for finishing earlier than the baseline compared to finishing later. This asymmetry is intentional, as finishing later already incurs additional penalties through the climate and complexity components of the reward function.
Furthermore, the implementation of the climate reward was modified by replacing the mean absolute error (MAE) with the mean squared error (Mean Squared Error

(MSE)) to more strongly discourage excessive changes in speed. The resulting climate reward for agent i at time step t is defined as:

$$r^i_{\text{climate},t} = \frac{1}{\text{cur_idx} - \text{prev_idx}} \sum_{k=\text{prev_idx}}^{\text{cur_idx}} \left[\left(v^i_{\text{actual},k} - v^i_{\text{baseline},k} \right)^2 + \left(h^i_{\text{actual},k} - h^i_{\text{baseline},k} \right)^2 \right].$$

Using the squared deviations increases the penalty for larger deviations from the baseline, thereby discouraging excessive changes in speed and encouraging the agent to follow the baseline trajectory more closely. For these changes to have a larger impact, the reward normalization was also de-activated.

5.3.2. Results

Figure 5.7 shows the mean action during evaluation episodes throughout training.

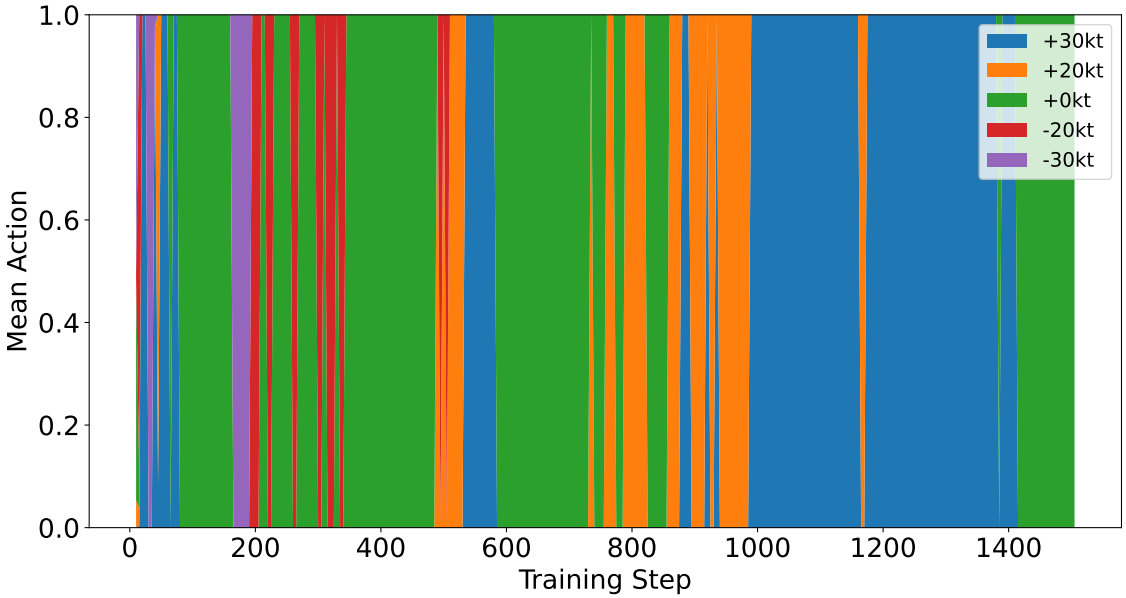


Figure 5.7.: Evaluation mean action throughout training

It can be observed that the adjustments made have a profound impact on the mean action. Most notably, the policy appears to no longer favor actions resulting in a speedup for prolonged periods of time.

However, it is still apparent that the resulting policy collapses to a single action, which is selected by all agents for the entirety of each episode. This indicates that a more fundamental issue is affecting the learning process. An analysis of the actor weights reveals that, during training, all weights of the *fusion* layer converge to zero, effectively cutting off the flow of information through the network. This phenomenon is characteristic of the *Dying ReLU* problem, in which neurons become inactive and cease to contribute to the network's output [44]. It is worth noting that this problem does not occur in the critic network. This is also reflected in the loss curves, where the actor loss becomes stuck while the critic loss continues to decrease. This behaviour can be seen in Figure 5.8 which shows the value and policy loss throughout training of the MARL agent.

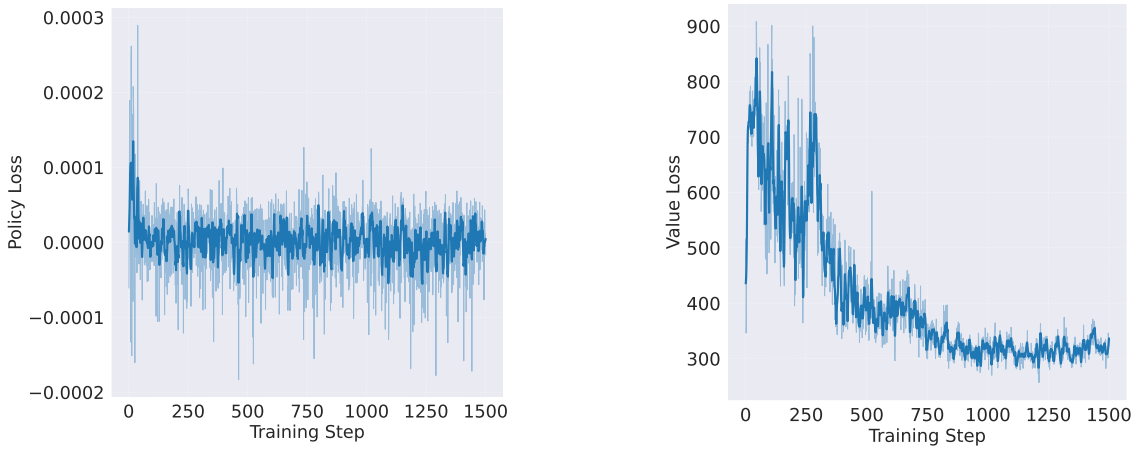


Figure 5.8.: Policy Loss (left) and Value Loss (right) throughout training

While the exact cause of this behaviour remains unclear, replacing the standard ReLU activation function with *Leaky ReLU* provides a potential approach to mitigating the problem. Small-scale experiments were therefore conducted using *Leaky ReLU* [45]. Although these experiments had not yet fully converged at the time of writing, their results indicate that the use of *Leaky ReLU* may prevent the observed policy collapse. Consequently, while no definitive conclusion can yet be drawn, the preliminary results suggest that replacing ReLU with *Leaky ReLU* is a promising approach to resolving the observed behaviour. Further experiments with longer training runs are required to confirm whether this effect persists after convergence.

6. Conclusion and future research topics

6.1. Conclusion

This thesis explored the application of reinforcement learning to climate-aware trajectory planning on a network scale. The primary focus were the engineering challenges in the design and implementation of a system to support future work on this topic. A system architecture comprised of a micro-scale trajectory optimization and a balancing phase using MARL, inspired by previous work, was adopted.
In the balancing phase, agents aimed to minimize a penalty composed of a complexity term and a climate term, where the climate term was calculated as the deviation from the ideal trajectory determined in the first phase.
In the micro-scale trajectory optimization, individual routes obtained from traffic data were optimized using existing tooling that incorporate accurate weather and aircraft performance data to determine the best route. These trajectories were subsequently adjusted in altitude and speed to meet the requirements of the MARL system.
A central part of this engineering effort was the development of a high-performance training environment. By combining a data-centered approach for storing the environment state with fully vectorized operations, the environment enables efficient parallel handling of a large number of agents.
The resulting MAPPO agents employ an actor network with two observation branches, processing the agent's own parameters as well as those of neighboring aircraft, and two output heads for the speed and altitude actions. A centralized critic estimats

the value function from a global observation, following the centralized-training-decentralized-execution paradigm.

Finally, two testing campaigns were conducted using the proposed architecture to evaluate the system components in an integrated setting. The first campaign served as an initial integration test and revealed reward-hacking behavior, with the policy converging toward a strategy that prioritized speed to reduce both the complexity and climate penalties. Based on these findings, a speed penalty was introduced for the second campaign to directly discourage this behavior.
The results of the second campaign indicated that the introduced speed penalty was effective in mitigating the previously observed tendency toward excessive speed. However, both campaigns also exhibited limited diversity in the policy's actions. A pronounced dying ReLU problem was observed in the actor network, which provides a plausible explanation for the limited diversity in the learned policy. While the effectiveness of alternative activation functions such as Leaky ReLU was not evaluated, the observed activation behavior indicates that the choice of activation function may be an important factor in the stability and expressiveness of the learned policy.

6.2. Future research topics

The system developed in this work provides a foundation for several promising directions for future research. The most immediate next step is to conduct further experiments to evaluate the approaches identified throughout the discussion. In particular, replacing the ReLU activation function with Leaky ReLU warrants further investigation, as it may help mitigate the observed dying ReLU behavior and improve the stability and expressiveness of the learned policy.

Another promising direction is the use of the precise climate reward implemented in this work. As this reward more accurately captures the underlying environmental impact, it may provide a stronger incentive for the policy to avoid undesirable strategies such as those discussed in Chapter 5. It is therefore hypothesized that the more accurate reward formulation could result in policies that better reflect the intended climate objectives. However, substantial improvements in computational efficiency may be required before effective learning with this reward becomes practical. In particular, developing a more computationally efficient and parallelizable method for estimating fuel consumption could significantly reduce the additional computational cost introduced by the precise reward.

Future research could also investigate the use of per-agent rewards instead of aggregating individual climate rewards into a single team reward. Providing each agent with an individual climate-related reward could improve learning by providing more direct feedback on how individual actions contribute to the overall objective. This could, in turn, alleviate the credit assignment problem associated with the climate component of the reward.

Finally, attention mechanisms [46] could be investigated as an alternative to zero-padding for handling the dynamic nature of the environment. Attention mechanisms may allow the policy to selectively focus on the most relevant agents and environmental features, providing a more expressive representation of variable-sized observations while avoiding some of the limitations associated with zero-padding.

Bibliography

[1] H. Lee and J. Romero. *Climate Change 2023: Synthesis Report. Contribution of Working Groups I, II and III to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change*. Geneva, Switzerland: IPCC, 2023, pp. 35–115. DOI: 10.59327/IPCC/AR6-9789291691647.

[2] World Meteorological Organization. *State of the Global Climate 2023*. Tech. rep. WMO-No. 1347. Geneva, Switzerland: World Meteorological Organization, 2024. URL: https://library.wmo.int/idurl/4/68835 (visited on 08/06/2026).

[3] Marina Romanello et al. "The 2023 report of the Lancet Countdown on health and climate change: the imperative for a health-centred response in a world facing irreversible harms". In: *The Lancet* 402.10419 (2023), pp. 2346–2394. DOI: 10.1016/S0140-6736(23)01859-7.

[4] Paulina Jaramillo et al. "Transport". In: *Climate Change 2022: Mitigation of Climate Change. Contribution of Working Group III to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change*. Ed. by P.R. Shukla et al. Cambridge, UK and New York, NY, USA: Cambridge University Press, 2022. Chap. 10. DOI: 10.1017/9781009157926.012.

[5] D. S. Lee, D. W. Fahey, A. Skowron, et al. "The contribution of global aviation to anthropogenic climate forcing for 2000 to 2018". In: *Atmospheric Environment* 244 (2021), p. 117834. DOI: 10.1016/j.atmosenv.2020.117834.

[6] Peter Berster, Marc Gelhausen, and Holger Pabst. *DLR Touristik Report 2026*. Tech. rep. Köln-Porz, Germany: Deutsches Zentrum für Luft- und Raumfahrt (DLR), Institut für Luftverkehr, 2026. URL: https://elib.dlr.de/224092/.

[7] Marc C. Gelhausen et al. "Results from the ATS-Level Assessment of the Clean Sky 2 Technology Evaluator". In: *Aerospace* 12.3 (2025), p. 185. DOI: 10.3390/aerospace12030185.

[8] Zarah Lea Zengerling and Volker Gollnick. "A comparative analysis of operational climate mitigation measures in aviation". In: *AIAA Aviation 2026 Forum*. 06/2026. DOI: 10.2514/6.2026-4157. URL: https://elib.dlr.de/225112/.

[9] Jessie R. Smith et al. "The climate opportunities and risks of contrail avoidance". en. In: *Nature Communications* 17.1 (01/2026). DOI: 10.1038/s41467-026-68784-8. URL: http://dx.doi.org/10.1038/s41467-026-68784-8.

[10] Hiroshi Yamashita et al. "Analysis of Aircraft Routing Strategies for North Atlantic Flights by Using AirTraf 2.0". In: *Aerospace* 8.2 (2021). DOI: 10.3390/aerospace8020033. URL: https://www.mdpi.com/2226-4310/8/2/33.

[11] A. Simorgh et al. "Robust 4D climate-optimal flight planning in structured airspace using parallelized simulation on GPUs: ROOST V1.0". In: *Geoscientific Model Development* 16.13 (2023), pp. 3723–3748. DOI: 10.5194/gmd-16-3723-2023. URL: https://gmd.copernicus.org/articles/16/3723/2023/.

[12] Fateme Baneshi, Manuel Soler, and Abolfazl Simorgh. "Conflict assessment and resolution of climate-optimal aircraft trajectories at network scale". In: *Transportation Research Part D: Transport and Environment* 115 (2023), p. 103592. DOI: https://doi.org/10.1016/j.trd.2022.103592. URL: https://www.sciencedirect.com/science/article/pii/S1361920922004187.

[13] Fateme Baneshi, María Cerezo-Magaña, and Manuel Soler. "Integrating Non-CO2 climate impact considerations in air traffic management: Opportunities and challenges". In: *Transport Policy* 155 (2024), pp. 274–286. DOI: https://doi.org/10.1016/j.tranpol.2024.06.023. URL: https://www.sciencedirect.com/science/article/pii/S0967070X24001860.

[14] Tabish Rashid et al. "QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning". In: *CoRR* abs/1803.11485 (2018). arXiv: 1803.11485. URL: http://arxiv.org/abs/1803.11485.

[15] Robert Walz Fabian Schwarz. "Identifikation von Herausforderungen und Einflussfaktoren bei der Anwendung von Reinforcement Learning Methoden auf autonomes Fahren in End to End Szenarien". 2026.

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

[17] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994.

[18] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG]. URL: https://arxiv.org/abs/1312.5602.

[19] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". en. In: *Nature* 518.7540 (02/2015), pp. 529–533.

[20] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347.

[21] RONALD J. WILLIAMS and JING PENG. "Function Optimization using Connectionist Reinforcement Learning Algorithms". In: *Connection Science* 3.3 (1991), pp. 241–268. DOI: 10.1080/09540099108946587. eprint: https://doi.org/10.1080/09540099108946587. URL: https://doi.org/10.1080/09540099108946587.

[22] Volodymyr Mnih et al. "Asynchronous Methods for Deep Reinforcement Learning". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 20–22 Jun/2016, pp. 1928–1937. URL: https://proceedings.mlr.press/v48/mniha16.html.

[23] Yukai Hou et al. "UAV Swarm Cooperative Target Search: A Multi-Agent Reinforcement Learning Approach". In: *IEEE Transactions on Intelligent Vehicles* 9.1 (2024), pp. 568–578. DOI: 10.1109/TIV.2023.3316196.

[24] Bizhao Pang et al. "A multi-aircraft co-operative trajectory planning model under dynamic thunderstorm cells using decentralized deep reinforcement learning". In: *Advanced Engineering Informatics* 65 (2025), p. 103157. DOI: https://doi.org/10.1016/j.aei.2025.103157. URL: https://www.sciencedirect.com/science/article/pii/S1474034625000503.

[25] Jakob N. Foerster et al. "Counterfactual Multi-Agent Policy Gradients". In: *CoRR* abs/1705.08926 (2017). arXiv: 1705.08926. URL: http://arxiv.org/abs/1705.08926.

[26] Peter Sunehag et al. "Value-Decomposition Networks For Cooperative Multi-Agent Learning". In: *CoRR* abs/1706.05296 (2017). arXiv: 1706.05296. URL: http://arxiv.org/abs/1706.05296.

[27] Christian Schröder de Witt et al. "Is Independent Learning All You Need in the StarCraft Multi-Agent Challenge?" In: *CoRR* abs/2011.09533 (2020). arXiv: 2011.09533. URL: https://arxiv.org/abs/2011.09533.

[28] Chao Yu et al. "The Surprising Effectiveness of MAPPO in Cooperative, Multi-Agent Games". In: *CoRR* abs/2103.01955 (2021). arXiv: 2103.01955. URL: https://arxiv.org/abs/2103.01955.

[29] Ritchie Vink et al. *pola-rs/polars: Python Polars 1.44.0*. Version py-1.44.0. 08/2026. DOI: 10.5281/zenodo.22078838. URL: https://doi.org/10.5281/zenodo.22078838.

[30] Fateme Baneshi, María Cerezo-Magaña, and Manuel Soler. "A deep multi-agent reinforcement learning framework for climate-aware aircraft trajectory planning considering air traffic complexity". In: *Knowledge-Based Systems* 332 (2026), p. 114826. DOI: https://doi.org/10.1016/j.knosys.2025.114826. URL: https://www.sciencedirect.com/science/article/pii/S0950705125018647.

[31] Abolfazl Simorgh and Manuel Soler. "Climate-optimized flight planning can effectively reduce the environmental footprint of aviation in Europe at low operational costs". In: *Communications Earth & Environment* 6.1 (01/2025), p. 66. DOI: 10.1038/s43247-025-02031-8. URL: https://doi.org/10.1038/s43247-025-02031-8.

[32] SVG Repo. *Plane SVG Vector*. CC0 License. URL: https://www.svgrepo.com/svg/500971/plane (visited on 08/13/2026).

[33] You-Dong Liang and B. A. Barsky. "A New Concept and Method for Line Clipping". In: *ACM Trans. Graph.* 3.1 (01/1984), pp. 1–22. DOI: 10.1145/357332.357333. URL: https://doi.org/10.1145/357332.357333.

[34] Alexander Kuenz et al. "Weather-Optimized Individual Routes in a Global Scenario". In: *2023 IEEE/AIAA 42nd Digital Avionics Systems Conference (DASC)*. 2023, pp. 1–8. DOI: 10.1109/DASC58513.2023.10311109.

[35] E. J. McShane. "Extension of Range of Functions". In: *Bulletin of the American Mathematical Society* 40.12 (1934), pp. 837–842.

[36] Hassler Whitney. "Analytic Extensions of Differentiable Functions Defined in Closed Sets". In: *Transactions of the American Mathematical Society* 36.1 (1934), pp. 63–89.

[37] Junzi Sun, Jacco Hoekstra, and Joost Ellerbroek. "OpenAP: An open-source aircraft performance model for air transportation studies and simulations". In: *Aerospace* 7.8 (07/2020), p. 104. DOI: 10.3390/aerospace7080104.

[38] CE Working Group on Complexity. *Complexity Metrics for ANSP Benchmarking Analysis*. 2006.

[39] Marc Shapiro et al. *pycontrails: Python library for modeling aviation climate impacts*. Version v0.63.3. 06/2026. DOI: 10.5281/zenodo.20836452. URL: https://doi.org/10.5281/zenodo.20836452.

[40] D.I.A. Poll and U. Schumann. "An estimation method for the fuel burn and other performance characteristics of civil transport aircraft in the cruise. Part 1 fundamental quantities and governing relations for a general atmosphere". In: *The Aeronautical Journal* 125.1284 (2021), pp. 257–295. DOI: 10.1017/aer.2020.62.

[41] Wei Hu, Lechao Xiao, and Jeffrey Pennington. "Provable Benefit of Orthogonal Initialization in Optimizing Deep Linear Networks". In: *CoRR* abs/2001.05992 (2020). arXiv: 2001.05992. URL: https://arxiv.org/abs/2001.05992.

[42] Haris Iqbal. *HarisIqbal88/PlotNeuralNet v1.0.0*. Version v1.0.0. 12/2018. DOI: 10.5281/zenodo.2526396. URL: https://doi.org/10.5281/zenodo.2526396.

[43] John Schulman et al. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. 2018. arXiv: 1506.02438 [cs.LG]. URL: https://arxiv.org/abs/1506.02438.

[44] Lu Lu et al. "Dying ReLU and Initialization: Theory and Numerical Examples". In: *Communications in Computational Physics* 28.5 (2020), pp. 1671–1706. DOI: 10.4208/cicp.oa-2020-0165. URL: http://dx.doi.org/10.4208/cicp.OA-2020-0165.

[45] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. "Rectifier nonlinearities improve neural network acoustic models". In: *Proc. icml*. Vol. 30. 1. Atlanta, GA. 2013, p. 3.

[46] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Appendix A.

Supplemental material

A.1. Complexity Reward Scaling

This section details both the characteristics of the complexity reward signal described in section 4.2.4 as well as the scaling approach that was ultimately implemented. As the airspace complexity naturally scales with the number of active aircraft in the environment a simple linear scaling was applied for early tests of the MARL system. While this did stabilize training enough for early policies to emerge qualitative analysis revealed significant instability in the complexity signal. To reslove this issue it is theorized that it is necessary to implement a different scaling. An in-depth data analysis is conducted to gain insight into the structure of the complexity reward signal and to identify a new scaling approach. For the analysis 20 spatial environments are simulated using a NOOP (no operation) policy and their trajectory data is stored. The simulation resulted in a total of 4418 data points for further analysis.

Figure A.1 shows the minimum, mean and maximum number of active flights in each step as bars. The minimum, maximum and mean airspace complexity of the environment is shown as a line plot.

It can be observed that the complexity signal is subject to significant fluctuations.

Further, it can be seen that the relative spread of complexity scores is higher than that of the number of active agents.

Figure A.2 confirms the intuition that the number of active aircraft and the complexity are highly correlated as they form a distinct cluster.

Using this insight a log-log linear regression (Power Law Fitting) can be performed to find a parameter β that satisfies the equation

$$x \approx \alpha * n_{\text{active}}^{\beta}$$

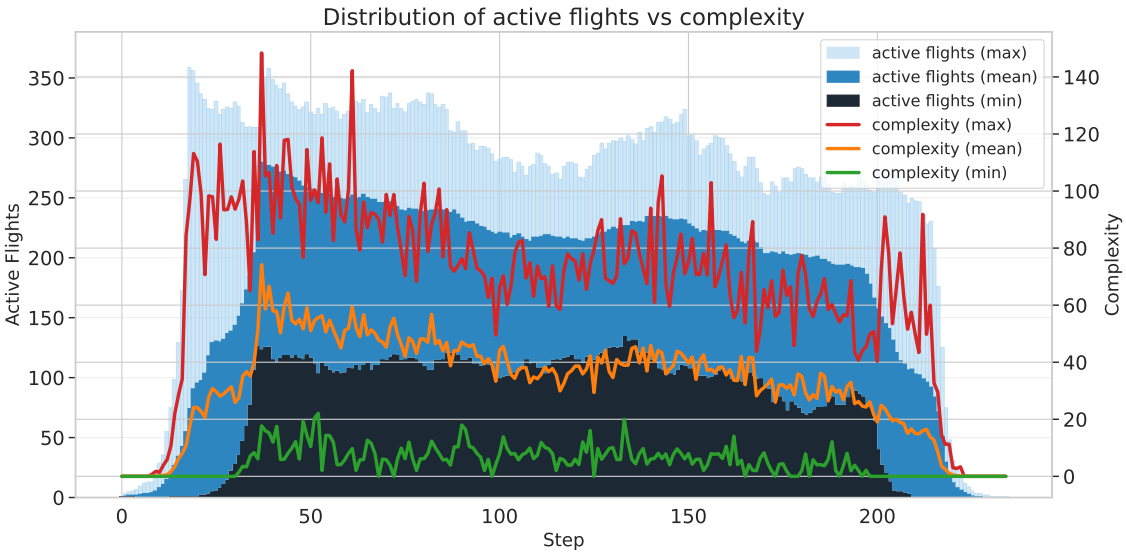


Figure A.1.: Distribution of active flights vs complexity

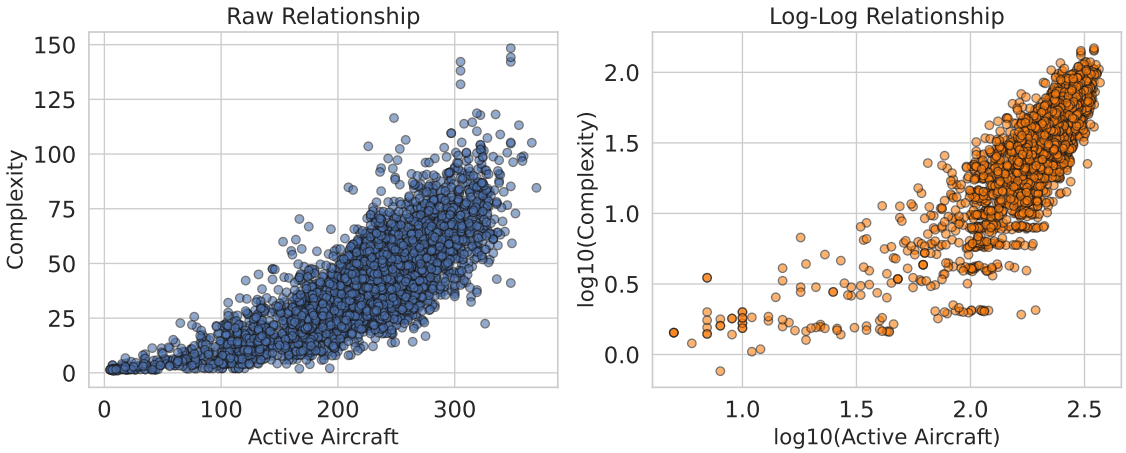


Figure A.2.: Correlation between number of active flights and complexity

where n_{active} is the number of active aircraft and α is a scaling coefficient that can be discarded in the final scaling.

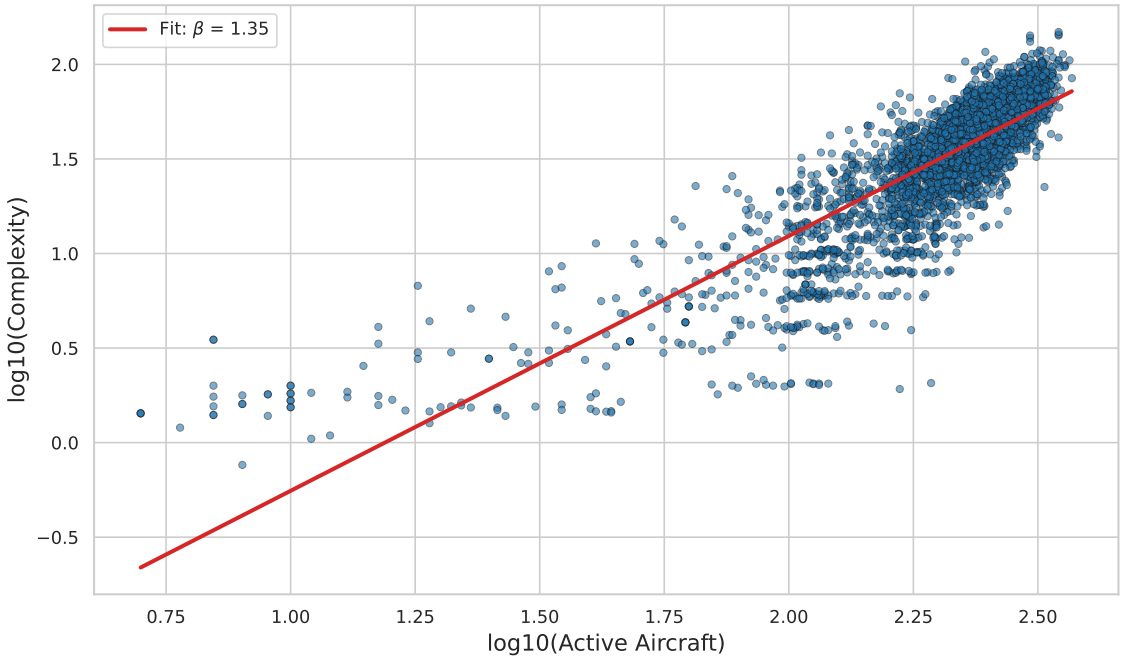


Figure A.3.: Power law fitting to find β

This fit yields an R^2 score of 0.702, indicating that approximately 70% of the variance in complexity is explained by the power-law relationship between the number of aircraft and complexity.
To measure the impact of this change on overall variance reduction the Earth Mover's Distance (EMD) metric is computed accross the dataset. The comparison between original, unscaled reward and the new scaling can be seen in figure A.4.

Overall, the nonlinear scaling with $\beta = 1.35$ reduced the average EMD between environments by 47% from 11.83 to 7.43 significantly reducing variance and therefore improving the reliability of the complexity signal. However; it can also be seen that variance remains relatively high. This is likely caused by the nature of the data as different regions may experience different traffic patterns.

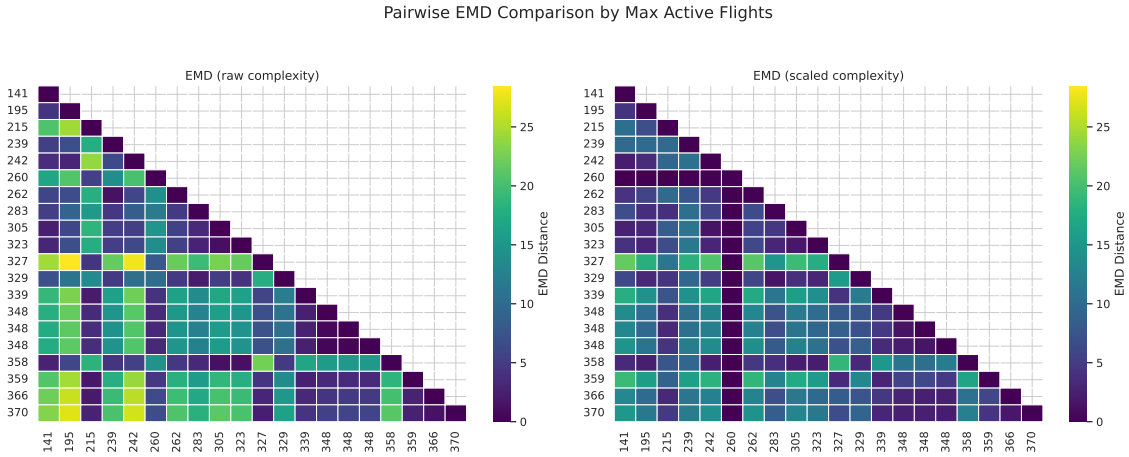


Figure A.4.: Pairwise EMD Comparison by Max Active Flights

A.2. Training data analysis

This section presents a short analysis on the training data used in section 5.2. The center points of the bounding boxes were generated by random in a bounding box of boundaries Figure A.5 shows the spatial distribution of the datasets used for training.

Table A.1.: Number of baseline trajectories and mean baseline length for each dataset.

Dataset	Number of trajectories	Mean trajectory length (N^b)
1	2035	17.62
2	1519	13.25
3	3549	14.72
4	3552	13.54
5	3191	15.04
6	3388	14.90
7	3536	14.01
8	2885	13.51
9	2736	15.99
10	1987	19.34

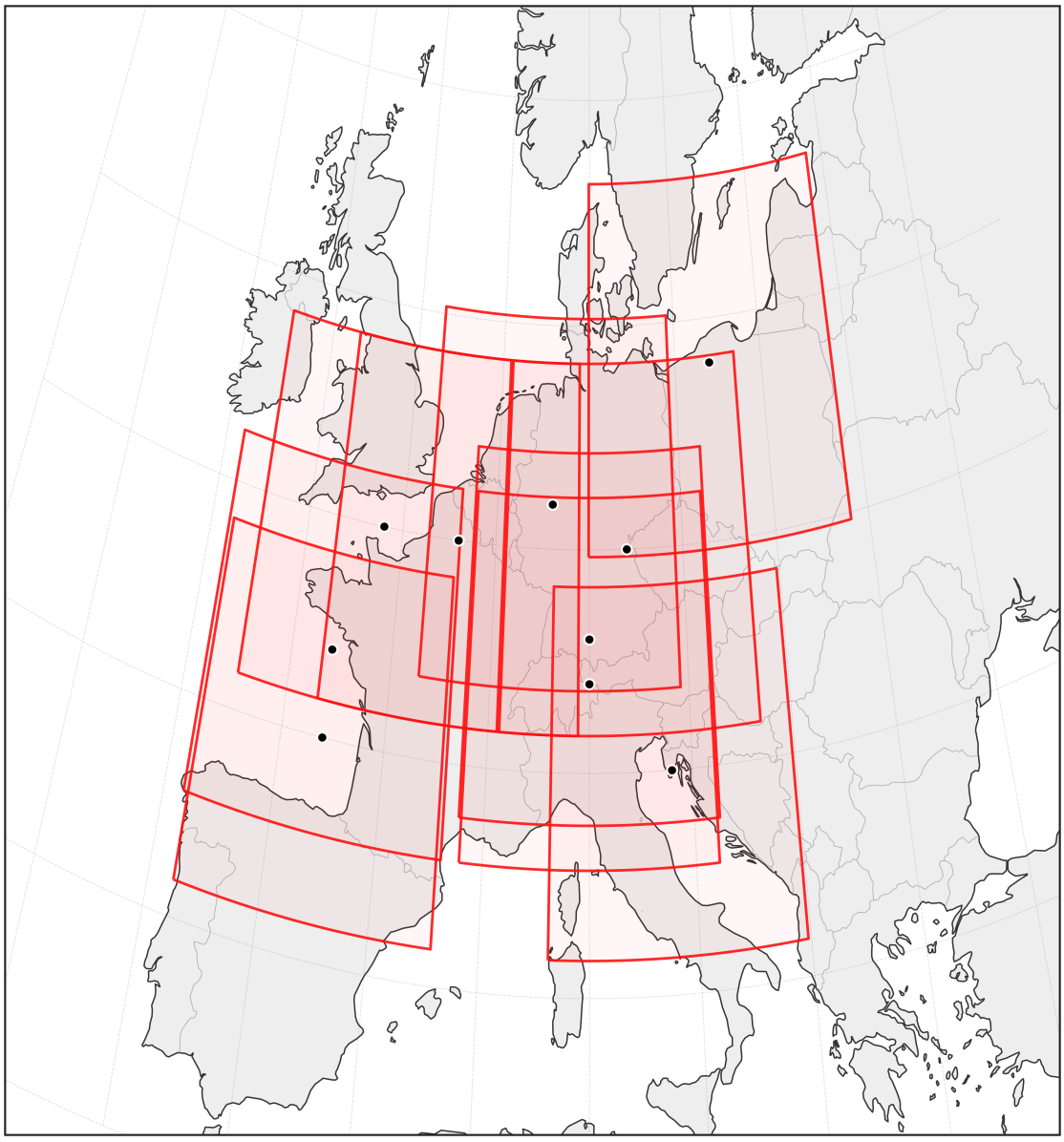


Figure A.5.: Spatial distribution of training datasets

A.3. State dataframe - columns & datatypes

Table A.2.: State dataframe

Column Name	Data Type
lon	`list<float64>`
lat	`list<float64>`
alt	`list<int32>`
dist	`list<int32>`
start_time	`datetime64[us]`
ground_speed	`list<float32>`
aircraft_type_pyBADA4	`string`
ICAO	`string`
flight_id	`uint32`
current_index	`int32`
previous_index	`int32`
segment_alts	`list<int64>`
segment_speeds	`list<float64>`
segment_times	`list[datetime64[ms]]`
current_alt	`int32`
current_speed	`float64`
active	`bool`
current_lon	`float64`
current_lat	`float64`
vert_mode	`int16`

A.4. Training Parameters

This section provides a quick reference for parameters used during training and testing of policies. Table A.3 presents the MAPPO-specific parameters used for the initial experiments

Parameter	Value
Minibatch Size	512
Epochs	1
Clipping Param (ϵ)	0.2
Value Coefficient c_1	0.5
Gradient Clipping Max	0.5
GAE gamma	0.99
GAE lambda	0.95
Entropy Coefficient Start	0.01
Entropy Coefficient End	0.005
Entropy Decay Steps	400
Bias (Linear Layers)	$\sqrt{2}$
Bias (Action Output Heads)	0.01
Actor Learning Rate	$1 * 10^{-3}$
Critic Learning Rate	$3 * 10^{-4}$
Optimizer	Adam

Table A.3.: MAPPO-specific Parameters

Environment Parameters:

Parameter	Value
Observation Box Side Length	40km
neighbor observation length	10
global observation length	400
initial aircraft count	50
maximum duration (step)	200
complexity scaling factor ($c_{\text{complexity}}$)	10
climate scaling factor (c_{climate})	$\frac{1}{500}$

The authors gratefully acknowledge the scientific support and HPC resources provided by the German Aerospace Center (DLR). The HPC system CARA is partially funded by "Saxon State Ministry for Economic Affairs, Labour, Energy and Climate Action" and "Federal Ministry of Research, Technology and Space".

This work was carried out in the context of the DLR ACCEPT project and is subject to the applicable project licenses and usage conditions. In particular, the DDR2 and BADA4 datasets used in this work were licensed within the framework of the ACCEPT project and are used in accordance with the terms and conditions of their respective licenses.