



UNIVERSITY OF
BIRMINGHAM

A PROOF OF CONCEPT FOR
AUTONOMOUS AIR TRAFFIC CONTROL
USING DEEP LEARNING

NOUR ASHOUSH



Under the supervision of DR. JIAN LIU

A thesis submitted to
the University of Birmingham
for the degree of Master of Engineering
in Computer Science and Software Engineering

School of Computer Science
College of Engineering and Physical Sciences
University of Birmingham
2024–25

ABSTRACT

This project presents a proof-of-concept system that demonstrates how deep learning can automate key aspects of ground-based air traffic control. A bespoke Python simulation of Birmingham Airport was developed, combining a realistic 2-D environment with route planning, aircraft physics and visual debugging tools. Three complementary learning components were integrated: deep Q-learning agents that taxi individual aircraft safely, a multi-agent reinforcement-learning framework that resolves surface conflicts through centralised training with decentralised execution, and computer-vision and natural-language-processing modules that offer runway occupancy monitoring and intent recognition from pilot requests. The system met its functional requirements by generating collision-free taxi routes, predicting aircraft movements, resolving taxiing conflicts and ensuring runway safety by monitoring occupancy, while running in real time on consumer hardware. Although the approach is constrained by simplified physics and limited training data, the results highlight the potential of data-driven, context-aware decision making to augment human controllers and lay the groundwork for future certification of AI-based decision support. The thesis contributes an open, modular simulation platform, a set of reproducible learning agents and an analysis of the practical, regulatory and human-factor challenges that must be addressed before autonomous ground control can transition from simulation to operational use.

ACKNOWLEDGEMENTS

First and foremost, I owe my deepest gratitude to my mum and dad for their unwavering belief in me. Their constant encouragement and quiet determination to see me succeed have been the foundation of everything I have achieved.

To my sisters, both of whom are brilliantly different from me, thank you for always hyping me up, even when you've had no interest in the things I was working on. Your support, despite not sharing my love for code and all things tech, has been a constant source of joy and motivation.

I'm also grateful to my supervisor, Dr Jian Liu, for giving me the freedom to run with my slightly wild ideas. His steady encouragement and trust let the work unfold in ways it simply couldn't have if there had been less freedom.

Finally, a heartfelt thank-you to the friends I've made at the University of Birmingham. Your chaotic all-nighters, motivational pep talks, and relentless banter kept me sane and made the hard work genuinely enjoyable.

ABBREVIATIONS

ADS-B	Automatic Dependent Surveillance-Broadcast
AI	Artificial Intelligence
AMAN	Arrival Manager
APM	Approach Path Monitoring
ASR	Automatic Speech Recognition
A-SMGCS	Advanced Surface Movement Guidance and Control System
ATC	Air Traffic Control
ATM	Air Traffic Management
ATMAS	Air Traffic Management Automation System
CFAR	Constant False Alarm Rate
CTDE	Centralised Training with Decentralised Execution
CV	Computer Vision
DL	Deep Learning
DMAN	Departure Manager
DQL	Deep Q-Learning
DSS	Decision Support System(s)
FAA	Federal Aviation Administration
GCRL	Graph Convolutional Reinforcement Learning
GPS	Global Positioning System
ICAO	International Civil Aviation Organization
ITAS	Integrated Tower Automation Suite
LSTM	Long Short-Term Memory
MARL	Multi-Agent Reinforcement Learning
MSAW	Minimum Safe Altitude Warning
NER	Named Entity Recognition
NLP	Natural Language Processing
RL	Reinforcement Learning
SMR	Surface Movement Radar
STCA	Short-Term Conflict Alert
TCAS	Traffic Collision Avoidance System
XAI	Explainable Artificial Intelligence
YOLO	You Only Look Once

GLOSSARY

Gate / Stand	Parking area at a terminal where aircraft load, unload and refuel.
Pushback	Backward tow, using a tug, to manoeuvre an aircraft off the stand.
Taxi / Taxiway	Low-speed ground movement between stand, runway and other areas; the paved route used.
Holding Point	Marked stop line where an aircraft must wait for clearance before entering a runway.
Runway Incursion	Any unauthorised presence of an aircraft, vehicle or person on a runway's protected area.
Take-off Roll	Accelerating along the runway until lift-off speed is reached.
Roll-out (landing)	Deceleration phase immediately after touchdown, before vacating the runway.
Go-around	Aborted landing: aircraft climbs away from the airport for another approach.
Runway Occupancy Time	Interval from an aircraft's wheels crossing the threshold until it vacates the runway.
Separation	Minimum safe distance (horizontal or vertical) mandated between aircraft.

TABLE OF CONTENTS

1	Introduction.....	1
2	Background and Motivation	2
2.1	The Complexity of Air Traffic Control	2
2.2	Existing Tools and Technologies in ATC.....	3
2.3	Limitations of Current Approaches	5
2.4	The Future Role of Artificial Intelligence in ATM	7
2.5	Challenges To Integrating AI in Operational ATC.....	9
2.6	Absence of Autonomous Decision-Making in ATC.....	10
2.7	Project Scope	11
3	Simulation Environment and System Setup	12
3.1	Overview.....	12
3.2	Functional and Non-Functional Requirements	12
3.3	Tools and Technologies Used.....	15
3.4	Airport Environment Design.....	15
3.5	Aircraft Modelling and Dynamics	19
3.6	Path Planning and Routing.....	22
4	Control of Aircraft Movement	24
4.1	Overview.....	24
4.2	Reinforcement Learning	24
4.3	Q-Learning.....	25
4.4	Deep Q-Learning (DQL).....	26
4.5	Implementation	27
4.6	Evaluation and Results.....	32
5	Aircraft Coordination and Conflict Management	34
5.1	Overview.....	34
5.2	Initial MARL Approach (CTDE).....	34
5.3	Introducing LSTM-Based Context Modelling.....	37
5.4	Reward Shaping and Experience Replay Techniques	40
5.5	Supervised Learning for Binary Decision-Making.....	45
5.6	Final Implementation and Trade-offs	47

6	Runway Monitoring and Management	48
6.1	Overview.....	48
6.2	Approach and Dataset Generation	48
6.3	Model Training and Performance	49
6.4	Integration into Runway Coordination Logic	50
7	Pilot Request Intent Recognition	52
7.1	Overview and Approach	52
7.2	Initial Implementation and Limitations	52
7.3	Dataset Preprocessing and Augmentation	53
7.4	Request Classification Using BERT.....	55
7.5	Flight Entity Identification and Candidate Ranking	55
7.6	Practical Experimentation.....	57
8	Project Management	58
8.1	Project Timeline.....	58
8.2	Development Methodology	59
8.3	Tools	60
9	Evaluation	61
9.1	Summary of Requirements	61
9.2	System-Level Behaviour and Findings.....	62
9.3	Component Evaluation.....	64
10	Conclusion and Discussion	65
10.1	Summary of Achievements.....	65
10.2	Limitations and Implications	65
10.3	Ethical, Legal and Safety Considerations	66
10.4	Future Work	66
11	References.....	68
12	Appendix.....	73

1 Introduction

Air traffic management (ATM) can be defined as the set of functions and systems needed to ensure the safe and efficient movement of aircraft in all phases of operation. This includes air traffic services, airspace management, and air traffic flow management. Within this framework, air traffic control (ATC) refers specifically to the service responsible for preventing collisions and maintaining orderly traffic flow of aircraft [1]. ATC is an important infrastructure that guides aircraft to move safely and efficiently in airspaces, which are becoming increasingly congested. It is responsible for handling aircraft in all phases of flight; departure, taxiing, take-off, en-route navigation, approach, and landing, which requires highly coordinated decisions to prevent conflicts and delays [2]. ATC operations are typically divided into three specialised domains: ground control, which manages aircraft and vehicles on taxiways and runways; tower control, which handles take-off and landing clearances; and area control, which supervises aircraft in transit between airports.

Across all these domains, it is human controllers that are responsible for the real-time decision-making, which they base on a wide array of variables such as aircraft positions, flight plans, weather data, and operational constraints. Their interventions and guidance are essential for maintaining safety by resolving conflicts and sustaining throughput. Unsurprisingly, the cognitive demands placed on controllers, especially during ground operations where traffic density is high, can exceed human limitations. Psychological studies have highlighted how even the most highly trained professionals are susceptible to performing errors due to attention lapses, memory constraints, and workload saturation [3, 4]. As air traffic continues to grow around the world, these limits are becoming a more pressing challenge for running airports safely and efficiently.

Technological aids like surveillance displays and alerting tools have been integrated to support controllers' decision-making, but these systems remain largely assistive in nature [5]. Most ATC procedures still rely on static rules and handcrafted heuristics that, while are effective under regular expected conditions, struggle to adapt unforeseen situations [3, 6]. In particular, the structured nature of ground movement and the availability of plenty surveillance data offer an underexplored opportunity for intelligent automation mechanisms. Despite the recent significant advances in artificial intelligence (AI), the use of data-driven and context-aware decision-making tools is not yet a standard feature of ATC systems [7].

This thesis therefore presents a proof-of-concept, limited to a virtual airport, intended as a sandbox for testing AI-driven intelligent models for ground control.

2 Background and Motivation

2.1 The Complexity of Air Traffic Control

ATC can be regarded as one of, if not the most, cognitively challenging operational roles in civil aviation. Despite the worldwide advancements made to airport infrastructures to optimise aircraft movement, and the introduction of many decision-support tools, the challenge of managing aircraft movements safely and efficiently continues to rely on human judgement. Air traffic controllers have the responsibility of maintaining situational awareness across a large, dynamic environment, and coordinating the safe operations of many aircraft in real-time. This responsibility continues to become increasingly complex as global air traffic volumes continue to rise.

Controllers usually work in airport control towers, which are buildings positioned to provide optimal visibility over runways and taxiways. Contrary to what is commonly assumed, controllers do not rely fully on radar displays for positional awareness, but instead, they have a mental model of aircraft locations and intentions, while using visual observations to validate and correct their mental assumptions [8]. However, the physical constraints of a tower's location often means that visual observations may be restricted, and this hinders the mental model that controllers develop. As airports grow and expand, terminal buildings and infrastructure often obstruct views, and aircraft located further from a tower can be difficult to monitor. Adverse weather conditions such as fog, heavy rain, or snow, can also significantly degrade the views from towers [9].

Although most major airports are equipped with essential technologies such as radar surveillance, instrument landing systems (ILS), and collision avoidance systems, each airport operates under unique constraints. No two airports share an identical operational layout, traffic profile, or climate. This means that these assistive systems are tailored on a case-by-case basis to meet the specific needs of a given airport, while still having to meet international safety regulations. On a global level, the International Civil Aviation Organization (ICAO) is the key figure in establishing and upholding aviation standards. Its Standards and Recommended Practices (SARPs) have been statistically proven to reduce accidents and improve safety. However, compliance with these standards varies across ICAO's 193 member states, which is usually due to infrastructural, economic, or governance limitations in each state [10].

Despite the regulatory frameworks placed to maximise the performance of ATC, there are many factors that hinder the performance of even the most well-trained controllers. As civil aviation grows, so does the number of reported incidents, many of which stem from cognitive overload or miscommunication. Safety regulations within aviation must be measurable and well-defined to avoid vague or inconsistent interpretations that may allow for the development of suboptimal systems, which in turn weaken organisational learning and long-term safety goals [11]. One area of growing concern is runway incursions, which is defined by ICAO as any unauthorised presence of aircraft, vehicles, or personnel on protected runway areas.

Between 2002 and 2020, over 26,000 runway incursions were recorded globally, and their occurrence rate is increasing faster than the growth rate of civil aviation itself [12].

Research shows that human error is a key factor in aviation incidents, with studies attributing approximately 70% of accidents to errors in human performance. The increased traffic volumes, tighter schedules, and the need to process large amounts of information leads to elevated workloads and impairs human judgement [4, 8]. Recent accidents highlight these vulnerabilities. For example, the Japan Transport Safety Board's (JTSB) interim report on the 2024 Haneda Airport collision detailed how a military aircraft entered a runway without clearance, and the ground controller failed to intervene in time to prevent a fatal collision with a commercial aircraft [13]. Similarly, in the 2025 Washington, D.C. mid-air collision, the National Transportation Safety Board (NTSB) found that ATC allowed separation between a military helicopter and a commercial jet to be done visually, but they did not enforce altitude restrictions which resulted in a mid-air collision [14].

These incidents show that even with the use of advanced surveillance and communication systems, the performance of ATC depends entirely on human performance. The compounding effects of limited visibility, workload saturation, and varying procedures creates situations where safety can be compromised. Usually, this is not due to negligence, but because the cognitive demand sometimes exceeds human limits [8]. This complexity raises the point to explore more adaptive, intelligent systems that can support human decision-making, particularly in the domain of ground control.

2.2 Existing Tools and Technologies in ATC

To manage the increasing complexity of air traffic operations, especially under high workload and low visibility conditions, a variety of systems have been developed to support controllers. These tools are made for the purposes of enhancing situational awareness, reducing cognitive load, and improving decision reliability. While implementations vary across airports, most systems fall into a few broad categories: surveillance, alerting, predictive, and decision support. This section looks at the existing assistive tools currently deployed in ATC, with a particular focus on their application to ground control.

2.2.1 Surveillance and Monitoring Systems

Surveillance systems are the foundation of modern ATC as they provide accurate, real-time information about aircraft and vehicle positions. These tools are essential for maintaining situational awareness, especially in complex or low-visibility conditions. By combining data from multiple sources, these systems help controllers track movements on the ground and in the sky, which help controllers ensure safe operations [15].

Surface Movement Radar (SMR) is a key surveillance tool used in ATC to monitor aircraft and vehicle movement on airport surfaces. It operates using radar waves to detect moving objects, with advanced signal processing techniques such as Moving Target Detection and Constant False Alarm Rate (CFAR) helping to distinguish genuine targets from noise and reduce false alerts [16]. The nature of using radar waves means it is especially useful in low visibility

conditions. SMR systems make use of larger surveillance networks, like GPS, to enhance its tracking accuracy and reliability [16]. These capabilities make SMR an essential tool for maintaining situational awareness in airport environments.

Advanced Surface Movement Guidance and Control Systems (A-SMGCS) is a more advanced system that builds on SMR by fusing data from various inputs such as radar, GPS, and video cameras, which is used to identify targets and provide automated routing and support. It has traditionally relied on statistical methods to perform this data fusion. However, the use of machine learning (ML) for data fusion and classification has recently been deployed to provide more accurate real-time information about aircraft and vehicle positions [17]. This system allows for better surveillance, control, and routing in a way that upholds safety standards.

2.2.2 Alerting and Safety Systems

Alerting systems play a vital role in air traffic safety by detecting potential conflicts and issuing prompt warnings to pilots or controllers. These tools monitor flight parameters to detect and prevent incidents such as mid-air collisions or deviations in altitude [18].

The Traffic Collision Avoidance System (TCAS) is one of the key airborne that help support safety during flight. TCAS is fitted on larger aircraft and is designed to prevent mid-air collisions. It works by sending out radio signals to nearby aircraft and receiving responses. This allows it to calculate distance, altitude, and relative movement. Based on this, it gives pilots audio alerts that suggest whether to climb or descend [19]. The algorithms behind TCAS are based on hardcoded logic and geometric calculations, although improvements are being researched to reduce signal congestion and improve accuracy through smarter filtering techniques [20].

Automatic Dependent Surveillance-Broadcast (ADS-B) is a GPS-based surveillance system that automatically broadcasts an aircraft's position, speed, and heading to ground stations and nearby aircraft. This system provides more frequent and precise updates than traditional radar and has become a core dependency of modern air traffic systems. Although ADS-B does not generate alerts on its own, it still enables alerting and decision-support systems due to its high accuracy and integration into other systems. For example, the basic version A-SMGCS does not include alerting functions, but more advanced configurations use ADS-B data and TCAS-like principles to generate safety alerts for ground movements [17, 20].

2.2.3 Predictive and Planning Tools

Predictive analytics in ATC involves the use of data-driven models to forecast traffic patterns, delays, and potential conflicts before they occur. These tools help controllers make decisions by analysing both historical trends and real-time data [15].

Arrival and departure management tools are used to help air traffic controllers plan the movement of aircraft into and out of busy airports. The Arrival Manager (AMAN) system supports sequencing of arriving aircraft by calculating their optimal arrival times based on landing rates, runway spacing, and other local constraints. This system's purpose is to reduce delays and improve runway usage while considering environmental impacts such as fuel

emissions. AMAN tools typically rely on *heuristic* methods. These are rules or strategies designed to find decent (not best) solutions for complex scheduling problems that cannot be solved in real time. A commonly used algorithm is *simulated annealing*, which searches through many possible landing sequences to find one that balances efficiency and fairness. Some AMAN systems also use ML models to predict delays and feed that information back to the planning algorithm [21].

Similarly, the Departure Manager (DMAN) focuses on managing departing flights. It plans take-off and engine start-up times to reduce taxi delays, avoid congestion, and coordinate with arrival schedules. Like AMAN, many DMAN systems began with rule-based or heuristic approaches but recently have been using more advanced optimisation techniques. For example, *genetic algorithms*, which are inspired by the process of natural selection, have been applied to improve the sequencing of departures and reduce overall waiting times [22]. Although the use of AI is still limited, both AMAN and DMAN show a clear trend towards combining traditional prediction methods with ML to improve predictive capabilities and decision-making.

2.2.4 Decision Support and Scheduling Systems

Decision support systems (DSS) are designed to assist air traffic controllers in making timely and informed decisions under high-pressure conditions. These systems are made to help resolve surface conflicts and maintain operational efficiency when visibility or workload is compromised. Typically, DSS provide real-time alerts, trajectory predictions, and sequencing recommendations to reduce reliance on human-based coordination [23].

In the Asia-Pacific region, the Air Traffic Management Automation System (ATMAS) serves as the regional standard for DSS. ATMAS is made up of a suite of modules such as Short-Term Conflict Alert (STCA), Minimum Safe Altitude Warning (MSAW), and Approach Path Monitoring (APM). These are integrated with surveillance data to help maintain aircraft separation and compliance with set procedures. The system supports trajectory prediction in 4 dimensions (where the 4th dimension is time) and can be integrated into customisable interfaces to better alert and aid controllers. Most ATMAS modules currently rely on rule-based logic and deterministic algorithms; for example, STCA uses geometric thresholding and look-ahead intervals for conflict detection. While ML is not yet operationally deployed, its modular architecture allows for future research and integration of AI such as classification for anomaly detection or alert prioritisation [15, 22].

By contrast, the United States employs the Integrated Tower Automation Suite (ITAS), which shares similar decision-support goals but lacks any publicly available technical documentation. Both systems reinforce a trend toward environments where AI-enhanced decision support is the future of these systems.

2.3 Limitations of Current Approaches

While a wide range of tools have been introduced into modern ATC to support situational awareness and decision-making, they remain limited in many key respects. These limitations

are not usually obvious due to the closed-source proprietary nature of some tools, but publicly available research and performance reports reveal that there are to extensive challenges across both the planning and control domains.

Despite the advancements made in scheduling and sequencing tools, many current systems still face constraints that limit their effectiveness. For instance, Extended Arrival Manager (E-AMAN) systems prioritise capacity optimisation over fuel-efficient routing, leading to increased fuel consumption and a higher constant workload for controllers. These kinds of systems also present feasibility challenges, since they demand high levels of real-time coordination, situational awareness from controllers and complex infrastructure to support the integration smoothly which some airports do not possess [21].

More broadly, current air traffic tools have been developed based on outdated design principles which were originally intended for older coordination and communication methods. These systems often lead to inconsistencies and poor decision-making, which need to be manually revised and fixed as situations evolve. As the industry shifts toward different systems, the accurate coordination of aircraft is essential, but many systems still lack the precision and synchronisation capabilities needed to fully implement advanced concepts [24].

The presence of human controllers the decision-making loop further complicates the application of these systems. Because human judgement is required for reliable planning and making context-aware decisions, any attempt to introduce some form of computational intelligence must account for the variability, delays, and errors introduced by manual intervention [22, 24]. Although simulations are used for training and testing management procedures, they still fail to capture or integrate the cognitive processes of controllers, resulting in systems that overlook the human factors and that negatively affects the decision qualities and increases workload to fix these decisions as the environment evolves [8, 22].

Another big issue is the limited real-time and predictive capabilities of current systems. The reliance on historical flight data and pre-defined flight plans means that these tools are not equipped to respond properly to unanticipated changes in traffic flow. In extreme environments like during weather disruptions or unexpected runway closures, the fixed-logic systems struggle to adapt and lose their efficiency [15]. Moreover, while decision-support tools can identify conflicts and present data visually, the actionable recommendations that they offer are scarce. The burden of resolving conflicts, sequencing traffic, and responding to alerts appropriately always remains on the controller, which leads to cognitive overload [8, 15].

All together, these limitations reveal the structural and functional constraints of current ATC technologies, and they highlight the need for more adaptive, context-aware systems capable of operating under uncertainty. The next generation of systems need to be able to provide controllers with cognitive support and respond intelligently to evolving or unseen traffic conditions.

2.4 The Future Role of Artificial Intelligence in ATM

2.4.1 Why AI Is Useful in ATC

As summarised in the previous section, a variety of helpful tools are used throughout ATC. However, their dependence on heuristic logic and rule-based algorithms limits their ability to generalise to unseen scenarios or adapt dynamically to different environments. Furthermore, attempting to modify or upgrade these systems often requires a complete redesign and revalidation of a system. While AI and deep learning (DL) have begun being integrated in ATC applications, the extent of their use is limited and is typically confined to small tasks such as data fusion [17]. Limiting AI to small tasks falls short of offering a more holistic application of AI within ATC systems. Nonetheless, the use of AI and DL in ATC is rapidly becoming the focus of ongoing research, with numerous experimental systems and concepts being proposed in academic literature.

AI has proven how valuable it is in domains where large volumes of high-dimensional data have to be processed in real time. In the context of ATC, it makes AI particularly well suited to analysing the complex and dynamic environments that would exceed the processing capacity of traditional rule-based systems. For instance, ML has been applied to risk assessment tasks by using meteorological and traffic incident data to create real-time conflict likelihood predictions, with accuracy of classification exceeding 92% [25]. These models can handle greater complexity than traditional heuristics and allow for adaptability as operations evolve.

From a more generic point of view, a major advantage of using AI and DL lies in its potential to scale and adapt better across different airspaces and operations. Once they are trained on representative datasets, AI models can be deployed more freely without needing reconfiguration, making them adaptable to global ATC environments. As traffic density and complexity continue to rise, it requires systems that can learn and generalise from data rather than rely on rules. If models are appropriately designed, they would be able to maintain performance across varying conditions and provide consistent support to controllers in regions of differing traffic. The flexibility that comes with AI has been recognised as the key motivator that is pushing the development of the next generation of air traffic systems [26].

2.4.2 Current AI Systems and Ongoing Research

One of the practical uses of AI within ATC is being implemented now at London Heathrow Airport, where AI is being used to improve runway efficiency and safety in low-visibility conditions. Eurocontrol [27] has reported that ML is being used to predict how long each aircraft will stay on the runway, making it possible to land planes closer together without compromising safety or overloading controllers. The main part of this setup involves using computer vision (CV) models like YOLO, which are used to detect and track aircraft in real time through video feeds. This visual data is combined with GPS information, giving a complete overview of aircraft positions. It means that even in poor weather, controllers have a clear view of what is happening on the ground. These technologies are part of a broader move towards “digital towers” or “virtual towers”, which combine AI with video feeds and traditional monitoring algorithms to improve decision-making. They have already proved to be useful

complex conditions like poor visibility or blocked sightlines. Importantly, all of this supports the shift toward AI-human collaboration, where AI helps people make better decisions rather than replacing them [28].

High-quality datasets are essential for training any AI systems to ensure models can properly interpret and respond to complex data. The ATCO2 corpus represents an advancement in the availability of ATC communication data for the development of speech-based AI systems. It was created through a collaborative effort involving open-source networks and multiple academics, and it contains over 5,000 hours of recorded pilot-controller communications. From this, a subset was manually transcribed by incorporating labels, call sign annotations, and detailed segmentation. The data reflects the linguistic diversity of international ATC exchanges, shown through the frequent presence of non-native English speakers and varied phraseology. The corpus served as the foundation for training domain-specific automatic speech recognition (ASR) models, which achieved word error rates as low as 17.9%. In addition, a call sign detection system was developed using the transcriptions, achieving precision rates above 90%, which enables AI tools to parse and act upon ATC instructions. As the first publicly available corpus of its kind, ATCO2 provides the necessary infrastructure for research in natural language processing (NLP) and automated decision support in ATC [29].

Recent research has focused on how NLP is being explored as a foundational capability for intelligent air traffic systems, since its use would establish context-aware communication interfaces. For instance, one research paper proposes a multi-stage NLP pipeline combining automatic speech recognition (ASR), named entity recognition (NER), and intent classification to transcribe and semantically label ATC commands [30]. Their system achieved a command recognition accuracy of 91.2% and intent classification F1 scores above 95%, demonstrating the promising future of the use of such pipelines. This growing interest in NLP-based understanding sets the foundation to integrate more autonomous reasoning systems, especially those driven by reinforcement learning (RL), which currently dominates ongoing research.

RL has gained attention in recent years due to its refinement through the training of large language models (LLMs) and its success across a range of decision-making applications. Recent developments have focused on RL's potential to support or automate tasks in ATC such as conflict resolution and cooperative control scenarios. A notable example Ghosh et al.'s [31] decentralised multi-agent system where each aircraft is treated as an independent RL agent that learns to adjust its speed and trajectory to minimise conflict, delay, and fuel use. Using realistic traffic simulations based on Eurocontrol data for evaluation, their approach was found to achieve better performance over traditional methods. Similarly, a graph convolutional reinforcement learning (GCRL) model has been shown to resolve conflicts between multiple aircraft while maintaining interpretable actions such as speed, altitude, and waypoint rerouting. The system was trained on real-world traffic data and successfully integrated into a Spanish ATC platform [32].

These systems show promise in their ability to adapt to various environments and generalise across scenarios, all while being able to support human interaction. Although many of these approaches remain at the simulation stage, they further reveal a paradigm shift toward data-

driven, adaptive control models that could eventually replace traditional ATC systems with ones that are context-aware.

2.5 Challenges To Integrating AI in Operational ATC

While the use of AI in ATC seems promising, its integration into day-to-day will likely be limited. The transition from simulation environments to real-world deployment is not straightforward, as several practical, organisational, and regulatory challenges will hinder the process. The following section outlines the key barriers that must be addressed before AI can be reliably incorporated into ATC systems at scale.

The biggest barrier to the AI's use in ATC is the lack of trust humans have which stems from the non-transparent nature of many AI systems. Despite the successes, such as the ones discussed in the previous section, current AI models often function as “black boxes,” and offer little transparency into how decisions or recommendations are formed. In safety-critical domains like ATC, this poses a big obstacle as controllers must be able to verify, interpret, and justify a system's outputs. The growing field of explainable AI (XAI) has emerged to address this issue by providing mechanisms to understand a model's behaviour and individual predictions. However, research has shown that even when explanations are provided, they do not always align with user expectations and can reduce trust further if inconsistencies are exposed [25, 26]. Ultimately, the absence of intuitive outputs limits AI's role in these systems and emphasises the need for ones that perform both well and are transparently communicative.

Equally important is the psychological impact of this lack of trust on human operators. When controllers feel uncertain about whether their tools will behave appropriately, they experience an increased cognitive workload and mental stress due to the need to constantly verify or override the system's actions. This heightened alertness places additional demands on the controllers, especially under time pressure [4, 6, 32]. To avoid this, AI systems must be designed to perform reliably, but with an emphasis on complementing and supporting controllers, rather than undermining them. This principle is already being followed in emerging frameworks such as the ASTAIR project, which explores AI-driven routing and conflict resolution in ground operations. Rather than replacing controllers, ASTAIR has AI act as a collaborative assistant, allowing for shared situational awareness and human intervention when necessary [7]. Given that human controllers will remain central to ATC for the foreseeable future, building systems that foster trust, and clarity is essential for effective future integration.

Another critical obstacle to the deployment of AI systems lies in the strictness of current certification processes and the broader regulatory environment. There are existing frameworks, such as DO-178C [33] for software assurance and ARP4754A [34] for system-level safety, that are built around deterministic, rule-based systems but are not well suited to the probabilistic nature of AI models. This discrepancy creates significant delays and uncertainties when attempting to validate AI systems for use in safety-critical domains. These traditional certification methods expect a system's behaviour to be fully specified and exhaustively verified, which is incompatible with models that evolve through learning or adapt in real time. As a result, even experimental AI systems that perform well in simulations are excluded from

operational trials due to the absence of regulatory pathways [25, 26, 35]. Recent efforts in the military aviation domain have begun to explore iterative and modular certification strategies for AI systems. They suggest incorporating criteria such as robustness, transparency, and adaptability into the certification itself, allowing for the partial approval of AI components in controlled contexts [35]. However, these methodologies require standardisations to be considered within civil aviation, where regulatory bodies slow the integration of novel technologies. Until these frameworks evolve to accommodate the characteristics of AI, its role in ATC will remain experimental.

A persistent challenge in advancing AI for ATC is the difficulty of achieving generalisability across different environments. Many learning-based systems are trained on narrow datasets or specific domains, meaning that they struggle to perform reliably in unfamiliar contexts. This is particularly problematic in ATC, where airspace configuration, traffic density, and controller preferences vary widely between regions. For instance, multi-agent reinforcement learning models have shown good results in simulations but often fail to generalise to new airspaces without extensive retraining or tuning of reward functions [20, 32]. Moreover, the way controllers behave and operate differs between regions, making it difficult to design AI systems that align with their various operational styles.

What adds to these limitations is the lack of standardised simulation environments and high-quality public datasets for training and evaluation. Current research efforts often rely on bespoke simulators and privatised datasets, making reproducibility performance comparisons difficult. As Wang et al. [20] highlight, there is no widely adopted baseline environment for evaluating conflict resolution algorithms in ATC, let alone domains outside of conflict resolution. This lack of infrastructure means collaborative progress is limited and development of deployable systems remains slow. Addressing this gap will be essential for moving AI research beyond prototypes and toward operational systems.

2.6 Absence of Autonomous Decision-Making in ATC

Despite recent advancements in AI and the increasing integration of ML into ATC systems, no frameworks exist which allow AI to perform autonomous decision-making. Instead, all AI applications within ATC to date have been restricted to the decision-support roles discussed earlier, providing only recommendations, alerts, and predictions that merely assist human judgement.

The distinction between decision-support and decision-making is critical. Decision-support systems (DSS) operate under the assumption that a human controller remains in the decision loop, who is responsible for interpreting outputs and making final the decisions. In contrast, decision-making systems would allow AI agents to act autonomously, potentially coordinating aircraft movements or resolving conflicts without any human oversight. To date, no such systems have been deployed in live ATC environments.

This lack of autonomous decision-making is not only because of the theoretical basis of the technology, but it is the regulatory constraints, lack of trust, and the absence of certifiable AI frameworks that causes the aviation industry to take very slow, cautious approaches to the

implementation of new technologies. Moreover, ATC operations remain highly diverse and with the wide variation in procedures, infrastructure, and safety culture across regions. These differences hinder the development of generalisable AI agents capable of managing the intricacies of real-world ATC environments.

Consequently, research into autonomous AI for ATC is both limited and theoretical. Until AI-based systems are widely trusted, certifiable, and consistent across international aviation systems, research efforts are likely to remain restricted to simulation environments. Nevertheless, these simulations provide an important testing ground for evaluating the feasibility of intelligent agents capable of collaboration and autonomous control.

2.7 Project Scope

This project offers a proof-of-concept, rule-based simulation in which reinforcement-learning taxi agents, a computer-vision runway-monitor, an LSTM trajectory predictor, and an NLP intent-classifier jointly issue ground-control clearances. All proposed models are confined to a simplified virtual version of Birmingham Airport, allowing them to act autonomously within the scenarios encountered during training and coded into the environment. It serves as a controlled sandbox for analysing how data-driven components might work together to coordinate aircraft movement without continuous human input.

Because every module is tuned to these limited virtual scenarios, the system cannot adapt to unseen traffic patterns, rare edge-cases or more sophisticated manoeuvres. It neither reasons adaptively nor satisfies the certification, robustness and interpretability requirements of operational ATC. The prototype should thus be seen as an exploratory demonstration of AI-enabled workflows which highlight both the promise and the immaturity of deep learning for the uses of ATC.

3 Simulation Environment and System Setup

This chapter presents the design and development of the simulation environment that acted as the foundation for training and evaluation of deep learning models to automate ground-based ATC. Given the experimental nature of this project, particularly in areas such as environment tuning, reward function design, and multi-agent coordination, a bespoke simulation was implemented from the ground up.

3.1 Overview

While there are existing open-source ATC simulators such as ‘BlueSky’, which are primarily used for training humans, they primarily focus on en-route traffic management and high-level airspace visualisation. They are not designed to support low-level, ground-based aircraft control at the level of taxiway navigation and runway coordination. Furthermore, integrating DL models into these platforms would introduce an added challenge of complexity and computational overhead. As such, a custom-built environment was essential to ensure flexibility, modularity, and alignment with the specific objectives of this project.

The simulation was developed entirely in Python to ensure compatibility with popular AI libraries. The environment itself was constructed using the Pygame library, which offers sufficient flexibility to implement real-time interaction, dynamic visualisation, and lightweight physics modelling. Although the simulation abstracts away many real-world complexities, it retains key operational elements such as taxiway navigation, runway coordination, and aircraft movement dynamics, making it a suitable platform to visualise various AI models in action.

This chapter begins by introducing the tools and technologies employed during development, followed by a detailed discussion of the airport layout, aircraft design, and system architecture. Particular attention is given to the design choices that balance realism with manageable complexity, and the rationale behind architectural decisions made during setup. This simulation environment forms the basis for the DL implementations described in later chapters.

3.2 Functional and Non-Functional Requirements

3.2.1 Functional Requirements of the Simulation Environment

The functional requirements for the simulation environment define the base capabilities necessary for modelling aircraft movement within an airport setting. These include loading from infrastructure from JSON files, route planning using Dijkstra’s algorithm, aircraft instantiation and movement, and a real-time visual interface. These features collectively form a somewhat realistic and modular environment to integrate and evaluate AI-driven ATC strategies.

ID	Requirement	Description
FR1	Airport Visualisation	The system shall load a visual representation of an airport as a background image.

FR2	Dynamic Infrastructure Loading	The system shall initialise stands, taxiways, runways, intersections, and holding points from a structured JSON file.
FR3	Layout Flexibility	The system shall support dynamic switching of airport configurations via editable image and JSON files.
FR4	Aircraft Instantiation	Aircraft shall be created as independent objects with their own properties, including position, heading, velocity, and assigned route.
FR5	Departure Simulation	The system shall simulate aircraft taxiing from a gate or stand to a runway for take-off.
FR6	Arrival Simulation	The system shall simulate aircraft taxiing from a runway to a designated stand or gate after landing.
FR7	Route Planning	The system shall use Dijkstra's algorithm to generate shortest-path routes through the taxiway network.
FR8	Movement Dynamics	Aircraft movement shall include realistic elements such as inertia, deviation from taxiway centrelines, and variation in speed.
FR9	Holding Point Recognition	The system shall define holding points at every runway entry or exit location where aircraft must stop until given clearance.
FR10	Visual Rendering	The environment shall display aircraft, taxi routes, and infrastructure elements using Pygame for real-time visualisation and debugging.
FR11	Model Access Interface	The environment shall allow for machine learning agents to send actions and receive environment observations.
FR12	Aircraft Isolation	By default, aircraft shall operate independently without collision detection or interaction awareness. Behavioural coordination is to be handled by external agents.

Table 3.A: Functional Requirements of the Simulation Environment

3.2.2 Functional Requirements of the Autonomous ATC System

The functional requirements of the autonomous ATC system outline the expected behaviours of the agents or models interacting with the simulation. These include managing clearances, resolving conflicts, enforcing runway safety rules, and prioritising aircraft based on wait time. An additional natural language processing (NLP) component is also defined to interpret and classify requests from aircraft, forming a practical human-system interface.

ID	Requirement	Description
ATC-FR1	Clearance Management	The system shall grant or deny clearance to aircraft for taxi, take-off, landing, or runway access based on current airfield state.
ATC-FR2	Conflict Avoidance	The system shall prevent multiple aircraft from occupying the same runway or conflicting taxiway segments simultaneously.
ATC-FR3	Runway Occupancy Control	The system shall ensure that only one aircraft is present on a runway at any time and shall allow sufficient time for aircraft to vacate before granting access to others.
ATC-FR4	Prioritisation Logic	The system shall prioritise aircraft that have been holding or waiting for extended periods when resolving movement conflicts.
ATC-FR5	Movement Prediction	The system shall anticipate aircraft trajectories based on current routes and status to proactively manage access and avoid deadlock.

ATC-FR6	Centralised Coordination	The controller shall coordinate actions across all aircraft centrally, though the architecture may also support decentralised alternatives.
ATC-FR7	NLP Request Interpretation	The system shall include an NLP component capable of inferring the intent and source of verbal or textual requests, classifying them as landing, take-off, taxi, or pushback requests.

Table 3.B: Functional Requirements of the Autonomous ATC System

3.2.3 Non-Functional Requirements

The non-functional requirements describe the qualities of the simulation and ATC system, such as real-time execution, scalability, realism in aircraft physics, modularity, and visual performance. These criteria make sure that the system remains suitable for research and experimentation while maintaining clarity and usability on standard hardware.

ID	Requirement	Description
NFR1	Real-Time Simulation	The simulation shall be capable of running in real time to ensure the data used for training and evaluation of models reflects realistic operational conditions.
NFR2	Variable Simulation Speed	The system shall offer a configuration to run at speeds faster than real time, depending on the machine's processing capability.
NFR3	Visual Performance	The simulation shall maintain consistent frame rates and visual fidelity to support real-time observation and debugging.
NFR4	Realistic Aircraft Movement	Aircraft movement shall adhere to simplified physical constraints, including proportional turn radius based on speed and restricted manoeuvrability at low speeds.
NFR5	Spatial Accuracy	The airport layout, taxiways, and aircraft models shall be to scale to maintain realism in distance, timing, and spacing.
NFR6	Modularity and Extensibility	The simulation shall be modular, allowing for new airport layouts, aircraft types, and model integrations via configuration files and minimal code changes.
NFR7	Configuration Management	A central configuration file shall be used to control simulation dynamics, parameters, and assets to facilitate experimentation and reuse.
NFR8	Code Maintainability	The codebase shall be structured in a way that supports maintenance, versioning, and future extensions. Basic documentation is included, with plans for further development before public release.
NFR9	Visual Debugging	The system shall include clear visual indicators (e.g. routes, positions, control points) to support model evaluation and debugging.
NFR10	Evaluation Support	The simulation shall support both visual inspection and quantitative metrics collection for performance analysis.
NFR11	Platform Compatibility	The system shall be developed in Python 3 and capable of running on standard consumer hardware with ≥ 8 GB RAM.

Table 3.C: Non-Functional Requirements for the Simulation and Control System

3.3 Tools and Technologies Used

The system was developed in Python 3.9 which was specifically selected as it was the most compatible with the required libraries and ensured runtime stability across platforms. Earlier versions lacked support for certain ML tools, while newer releases introduced compatibility issues, particularly when using hardware acceleration such as the Metal Performance Shaders (MPS) backend on macOS systems.

Several Python libraries were employed across the different components of the system. NumPy [36] was used for handling numerical operations. PyTorch [37] served as the primary DL framework for neural network-based architecture. For computer vision components, OpenCV [38] and Ultralytics [39] were used to support object detection and visual inference. Additional libraries such as pandas [40] and scikit-learn [41] were used for dataset handling, preprocessing, and conventional ML experiments. The Transformers library [42] was incorporated to support pre-trained models used in natural language processing and understanding tasks.

All neural networks were trained using the Adam optimiser [43], which adaptively adjusts learning rates by estimating the first and second moments of the parameter gradients. Additionally, the ReLU (Rectified Linear Unit) activation function was used in all hidden layers due to its speed and effectiveness. It is defined as $f(x) = \max(0, x)$ and it adds non-linearity to a neural network without affecting computational performance due to its simplicity.

The 2D simulation environment was built using Pygame [44], a lightweight rendering and event-handling library. Pygame was chosen for its simplicity and speed in rendering 2D environments, making it particularly suitable for simulating ground-based air traffic control from a top-down perspective. Visual outputs were essential for debugging and evaluation, and Pygame provided a sufficient level of graphical control without the overhead of a more complex and hardware intensive engine.

Development was conducted primarily in Visual Studio Code, with Jupyter Notebooks used for certain cases of model training, prototyping and evaluation. The project was structured around a central simulation environment, which served as the integration point for trained models. Each DL component was developed within its own self-contained environment. Once trained and validated, models were integrated into the main simulation to enable evaluation in global conditions. This modular approach allowed for experimentation with different architectures without disrupting the integrity of the central system.

3.4 Airport Environment Design

A critical aspect of the simulation environment was the ability to dynamically generate airports based on external data. This flexibility was essential to support a wide range of experimental scenarios and to ensure that the system could be easily adapted for future expansions. To enable this, the airport layout was designed to be generic and allow any airport to be visualised and simulated within the environment, provided the relevant input data was supplied.

Using Pygame for the simulation window, a fixed rendering resolution of 1200 pixels in width and 600 pixels in height was maintained throughout. These dimensions were chosen to reflect the typical aspect ratio of a modern display and to allow for a horizontal, radar-style view of the airport environment. This resolution also provided a practical balance between performance and visual clarity, ensuring that the simulation could run smoothly on standard hardware while maintaining sufficient screen space to represent key airport elements such as taxiways, stands, and runways.

For the proof-of-concept, Birmingham Airport (EGBB) was selected as the reference airport. The decision to use Birmingham instead of a larger international airport such as Heathrow was motivated by several factors:

- **Simplified Layout:** Birmingham Airport features a clean and straightforward layout, ideal for initial prototyping and reducing cognitive overhead.
- **Single Runway Operations:** The airport operates a single runway for both take-offs and landings, which simplifies conflict management and makes it ideal for testing shared-runway logic.
- **Balanced Complexity:** The airport is large enough to simulate meaningful scenarios but avoids the overwhelming scale and congestion of major international airports.
- **Visual and Structural Intuition:** The layout lends itself well to simulation, making it easier to develop, visualise, and explain key concepts in the system.

A high-resolution image of the airport was extracted from Google Earth. Rather than capturing the image in a standard north-up orientation, it was deliberately 60° anticlockwise rotated to create a horizontal alignment of the runway. This decision ensured that the runway could be displayed cleanly across the full width of the simulation window, enhancing readability and making better use of screen space. To further improve the quality and realism of the background, the image was manually cleaned using Adobe Photoshop. This involved removing visible aircraft, shadows, and clutter such as map annotations. The goal was to provide a clean, distraction-free view of the airport layout, suitable for both simulation and visual debugging.



Figure 3.A: Cleaned and pre-processed image of Birmingham Airport used as the simulation background, aligned horizontally to maximise visual space.

Once integrated into the simulation, the 3-kilometre runway spanned approximately 1170 pixels in width. This established an approximate in-simulation scale of 1 pixel ~ 2.5 metres,

which was later used in computing realistic aircraft speeds, distances, and movement dynamics [45].

To accurately map the airport layout in terms of navigational structures such as taxiways, holding points, and stands, the official ICAO Aerodrome Chart for Birmingham Airport was consulted [46]. This chart provided a detailed reference for manually defining the airport geometry in Python. The extracted data formed the basis for creating the JSON file used by the simulation to render and structure the airport, ensuring that the layout corresponds to the real-world configuration and is easily interpreted by the system.

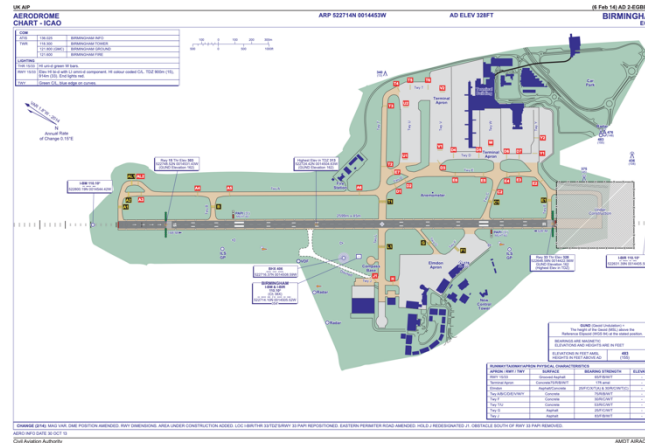


Figure 3.B: ICAO Aerodrome Chart for Birmingham Airport (EGBB), sourced via OpenNav [46], showing airport layout.

To enable a modular and data-driven airport environment, the layout of the airport was represented in a JSON file. This approach disassociates the airport's design from having the logic of the remainder of the system, allowing different airports to be loaded up by simply altering input data.

The JSON file defines several key components:

- **Nodes:** Represent navigation points in the airport, such as intersections or holding positions. Each node is uniquely identified by an ID and has associated x, y pixel coordinates corresponding to locations in the airport.
- **Edges:** Define directed taxiway segments connecting two nodes. These serve as the basis for route planning and aircraft path-following.
- **Stands:** Represent parking positions for aircraft, defined by their location, heading, and their associated parent taxiway (used for parking logic).
- **Holding Points:** A subset of nodes where aircraft must obtain clearance before proceeding, particularly before entering or exiting a runway.
- **Runway Exits:** Enumerate valid exit paths for aircraft landing on either end of the runway.

Before describing the process of loading airport layout data from JSON, it is helpful to outline the core data structures used in the simulation. The Node, Edge, and Stand classes form the basis for modelling taxiway networks, connections, and aircraft parking locations. Figure 3.C summarises the attributes of each class.

Node	Edge	Stand
id: str x: int y: int holding_point: bool	id: str start: Node end: Node distance: float heading: int	id: str x: int y: int heading: int parent: Edge available: bool

Figure 3.C: Summary of core classes used to model an airport's infrastructure.

During simulation initialisation, the `initialise_graph()` function is invoked to parse this JSON structure. This function performs several steps:

1. **Node Loading:** Nodes are stored in both a dictionary (NODE_DICT) for direct access by ID and a list (NODE_LIST) for iteration. Each node is instantiated as a Node object, and if its ID appears in the list of holding points, the `holding_point` flag is set.
2. **Edge Construction:** Edges are instantiated between connected nodes with both forward and reverse directions, enabling bi-directional traversal. Each Edge object contains the computed distance and heading based on the coordinates of the connected nodes. These are stored in both a list (EDGE_LIST) and a dictionary (EDGE_DICT) for efficient lookup and processing.
3. **Stand Parsing:** Each stand is defined as a Stand object, linked to a parent edge, and stored in both STAND_LIST and STAND_DICT. This allows aircraft to spawn at or return to specific parking positions with correct orientation and edge alignment.
4. **Runway Exit Mapping:** Runway exits are stored for each direction of travel, defining how aircraft can safely exit the runway upon landing.

This separation between layout data and simulation logic enables full generalisation. Any airport can be simulated by providing a corresponding image and JSON layout. The use of both list and dictionary structures gives the simulation more flexibility when coding the logic.

To render the airport layout on top of the background image, a `render_airport()` function was implemented. This function overlays all key components using Pygame's drawing functions. Each element is rendered conditionally, allowing for selective visualisation depending on the need. Nodes are drawn as red circles, each accompanied by a label of its unique ID. Edges connecting the nodes are drawn as lines, and stands are displayed as blue squares representing parking positions.

It is important to note that, contrary to standard Cartesian intuition, Pygame's coordinate system originates from the top-left corner of the display window. As such, increasing the y-coordinate moves downward on the screen, effectively flipping the y-axis compared to typical mathematical conventions [44]. While this inversion may cause trigonometric calculations (e.g., sine and cosine) to appear counterintuitive, this had no impact on the system's correctness.

Although the simulation does not use real-world taxiway naming conventions, the node and edge identifiers were inspired by the structure of real-world ground navigation systems. Names such as "A1" and "T2" mimic the alphanumeric formats often seen in taxiway networks, where

a letter denotes a route, and a number enumerates a segment. This approach provides clarity and consistency without requiring exact alignment with ICAO naming standards [46].



Figure 3.D: Visual rendering of a subsection of the airport layout showing nodes, edges, and stand positions overlaid on the background image.

In summary, the airport environment was designed to be flexible, modular, and visually informative. The combination of real-world references, structured data representation, and graphical rendering provides a foundation for simulating somewhat realistic ground operations and integrating intelligent decision-making models.

3.5 Aircraft Modelling and Dynamics

Each aircraft in the simulation is modelled as an independent object, instantiated from a custom Plane class. This object-oriented approach allows each plane to maintain its own state, trajectory, memory, and behavioural attributes throughout its lifecycle.

Visually, aircraft are represented using a 17×17 -pixel sprite image created in Adobe Photoshop, and their position and orientation are rendered in real time on the simulation background. The dimensions of the aircraft sprite (17×17 pixels) were chosen based on the scale of the background image (shown in Figure 12.A in the Appendix). At a scale of approximately 2.5 metres per pixel, this corresponds to a real-world aircraft length and wingspan of around 42.5 metres, roughly matching the size of a Boeing 737, one of the most common commercial aircraft globally [47].

For the sake of simplicity, only a single aircraft type is modelled within the simulation. While Birmingham Airport accommodates a wide variety of aircraft with different sizes and performance characteristics, all aircraft in the simulation share the same physical dimensions and dynamic properties [45]. This decision helps maintain consistency in behaviour modelling and avoids unnecessary complexity in early-stage prototyping. Each plane has associated attributes including position (x, y), heading angle (in degrees), speed, and thrust setting. During initial prototyping, these values were manually adjusted using keyboard input to develop an intuitive sense of how an aircraft should respond to directional changes and inertia.

Planes begin their lifecycle in a preparing state, typically parked at a designated stand, before transitioning into active operational states such as taxiing or takeoff. Each aircraft object also tracks a priority value used in conflict resolution and an authority flag used to manage

clearance status. These attributes become increasingly important when control is handed over to learning-based agents.

Each aircraft has its own state variable that represents its current phase of operation. This state information is essential for ensuring context-appropriate behaviour. The full list of operational states includes:

- preparing: Waiting at the stand until departure preparations are complete
- pushback: Reversing from the stand and reorienting for taxi
- taxiing: Moving along a planned route toward the runway
- takeoff: Aligned on the runway and accelerating with full thrust
- landing: Descending and touching down at a defined runway entry
- rollout: Slowing down after landing and exiting the runway
- parking: Navigating to the designated stand for parking
- parked: Stationary at the gate or stand
- go_around: Abort landing and exit airport perimeter

A visual state diagram is included in Figure 3.E to show the transitions between these phases during a typical arrival or departure sequence.

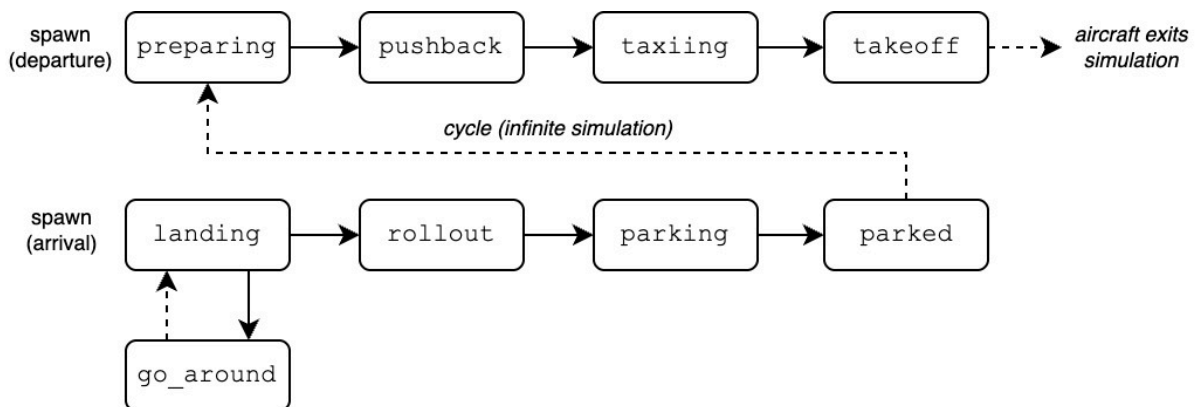


Figure 3.E: A state transition diagram of an aircraft's operational states

Aircraft pushback, take-off, and landing procedures are hardcoded, since training separate AI models for those tasks is irrelevant for the scope of this project. For take-off, an aircraft is positioned and aligned with the runway centreline, and its thrust is set to 100%. For landing, an aircraft is spawned at the end of the runway, properly aligned with its direction of travel. During rollout, the plane continuously evaluates which runway exits it has passed, using the numeration from the runway_exits JSON field to identify the next viable exit point. Pushback is simulated by reversing the aircraft briefly and orienting it toward the appropriate heading to begin taxiing.

Aircraft movement is simulated using simplified physical dynamics intended to balance realism with computational efficiency. These dynamics control how aircraft accelerate, decelerate, and turn, both during taxi operations and higher-speed phases such as take-off and landing. Aircraft speed is updated each frame based on applied thrust, capped by a maximum acceleration rate. Simultaneously, a drag coefficient acts as a resistance force, reducing speed

proportionally. This models the effect of ground friction or air resistance. The speed update equation is given as:

$$v_{t+1} = v_t + \left(\frac{\tau}{100} \cdot a_{\max} - \mu \cdot v_t \right) \cdot \Delta t$$

where v_t is the current speed, τ is the thrust percentage, $a_{\max}$ is the maximum acceleration, μ is the drag coefficient, and Δt is the time step per frame.

The aircraft's position is updated based on its current heading and speed. Assuming the heading is in degrees, the plane moves according to:

$$x_{t+1} = x_t + \frac{v_t}{\alpha} \cdot \sin(\theta) \cdot \Delta t$$

$$y_{t+1} = y_t - \frac{v_t}{\alpha} \cdot \cos(\theta) \cdot \Delta t$$

where θ is the heading angle, and α is the pixel-to-metre ratio derived from the airport layout.

Turning behaviour is modelled such that the turn angle decreases with increasing speed, mimicking the physical limitation of tight turns at high velocity. The turning angle per frame is calculated using:

$$\delta = \frac{\theta_{\max}}{1 + k \cdot v_t}$$

where $\theta_{\max}$ is the maximum turn angle, and k is a configurable scaling parameter. This relationship results in a wider turning radius at higher speeds, as illustrated in Figure 3.F, which compares the curvature of aircraft paths at different velocities. To further enhance realism, no turning force is applied when the aircraft is stationary, meaning that the plane must be in motion to turn. This constraint ensures accurate behaviour and prevents agents from learning unrealistic policies such as rotating on the spot while idle.

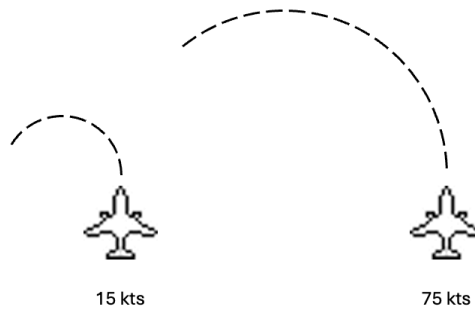


Figure 3.F: Visual illustration of aircraft turning behaviour at different speeds. At higher speeds, aircraft follow wider arcs due to reduced turning responsiveness.

These dynamics were tuned empirically during early development. A single aircraft was manually controlled using keyboard inputs to evaluate turning radius, acceleration response, and braking realism. Through trial and error, parameters were refined to reflect realistic taxi speeds and smooth transitions between acceleration and deceleration phases.

The simulation distinguishes between taxiing and take-off/landing modes by adjusting the drag parameter. On the ground, drag simulates rolling resistance, while in the air, reduced drag allows acceleration to realistic take-off or landing speeds. These states are switched by updating the drag attribute based on the aircraft's operational mode.

All simulation constants controlling aircraft motion, such as acceleration rate, drag coefficients, maximum speed, and turn rate, are centralised in a shared configuration file (`configuration.py`). This design allows consistent parameter usage across all simulation modules and enables easy tuning of motion dynamics without modifying the core codebase.

Config Variable	Symbol	Description	Value
MAX_SPEED	$v_{\max}$	Maximum aircraft velocity (used as upper cap on speed)	150 m/s
MAX_ACCELERATION	$a_{\max}$	Maximum acceleration when thrust is at 100%	5 m/s ²
GROUND_DRAG	μ	Drag coefficient when aircraft is taxiing	0.25
TAKEOFF_DRAG	μ_{airborne}	Drag coefficient during take-off roll	0.06
LANDING_DRAG	μ_{landing}	Drag coefficient during landing deceleration	0.04
MAX_TURN_ANGLE	$\theta_{\max}$	Maximum allowed change in heading per frame	1°
TURN_SPEED_SCALING	k	Scaling factor reducing turn rate at higher speeds	0.05
TIME_STEP	Δt	Time per simulation frame (based on 30 FPS)	$\frac{1}{30}$ s
PIXEL_TO_METRE	α	Conversion ratio between pixels and metres in the simulation	2.5 m/pixel

Table 3.D: Simulation Constants Used for Aircraft Motion.

Although all physical calculations in the simulation are performed in meters per second (m/s), aircraft speeds are displayed in knots to reflect standard aviation practice. This conversion is applied only at the visual display level and does not affect the simulation's internal consistency or physical modelling¹.

3.6 Path Planning and Routing

To determine the optimal route for an aircraft to taxi between two points, such as from a stand to a runway or from a runway to a gate, the simulation uses Dijkstra's algorithm [48] to compute the shortest path within the airport's graph structure. The airport is represented as a graph composed of nodes (waypoints and intersections) and edges (taxiway segments), each with an associated distance. Dijkstra's algorithm is a classic shortest path algorithm that guarantees the shortest path in graphs with non-negative edge weights. It systematically explores the graph from a start node, updating the shortest known distances to all other nodes, and selecting the next node with the lowest cumulative distance until the goal is reached.

¹ 1 m/s = 1.94384 knots

In this simulation, each node is uniquely identified, and edges between them are defined in the JSON layout file. During route generation, the `dijkstra(start_id, end_id)` function is called to return both the shortest path as a list of node IDs, and the total distance of that path.

The choice to use Dijkstra’s algorithm over alternatives such as A* [49] was informed by both the structure of the airport graph and practical considerations. While A* typically offers superior runtime performance in large or complex graphs by leveraging heuristics, such heuristics are not always intuitive or transferable across diverse airport layouts, but in fact may introduce noise in smaller graphs due to its heuristic nature, which might not be necessary or beneficial in simpler graph layouts. In contrast, Dijkstra’s algorithm guarantees optimality in weighted graphs with non-negative edge weights, and it provides more stable and predictable results in smaller, structured graphs like airport taxiway networks [50].

```
function DIJKSTRA(start_node, end_node)
    for each node in graph:
        distance[node] ← ∞
        previous[node] ← None
    distance[start_node] ← 0
    priority_queue ← [(0, start_node)]

    while priority_queue not empty do:
        (current_dist, current_node) ← pop lowest from priority_queue
        if current_node = end_node:
            break

        for each (neighbour, edge_cost) of current_node:
            new_dist ← current_dist + edge_cost
            if new_dist < distance[neighbour]:
                distance[neighbour] ← new_dist
                previous[neighbour] ← current_node
                push (new_dist, neighbour) to priority_queue

    path ← []
    current ← end_node
    while current is not None:
        path.append(current)
        current ← previous[current]

    return reverse(path), distance[end_node]
```

Algorithm 1: High-level pseudocode for Dijkstra’s algorithm used for shortest-path taxi routing between nodes.

Once a path is calculated using Dijkstra, it is assigned to the aircraft as a sequence of waypoints representing the shortest navigable route from a given start node (e.g. stand or runway entry point) to an end node (e.g. runway threshold or arrival gate). This path is stored in the plane’s path attribute. As the aircraft progresses, each visited waypoint is removed from the path and appended to the visited list, allowing the plane to keep track of its current position and dynamically determine the next node to approach.

4 Control of Aircraft Movement

4.1 Overview

To simulate an airport environment in which aircraft operate as autonomous entities, it was essential to move beyond static, rule-based logic and toward a framework that supports decentralised decision-making. Traditional rule-based systems are sufficient for deterministic path-following, but they lack the flexibility to adapt to changing environmental conditions or independent agent objectives. By contrast, a learning-based approach allows each aircraft to act as an intelligent agent that can observe its surroundings and take actions based on what it sees.

This section presents the design and implementation of such learning-based agents using reinforcement learning (RL). Each aircraft was modelled as an independent RL agent tasked with navigating the taxiway network autonomously. This not only introduced dynamic and varied behaviour into the simulation but also allowed the system to recreate the variability and unpredictability observed in real-world aircraft movements.

The decision to use autonomous agents was motivated by both simulation and real-world considerations. Within the scope of this project, learning-based agents allow for more realistic aircraft movement patterns and create opportunities to test coordination strategies. However, the broader use-case of this approach lies in its potential application as a verification and monitoring tool in air traffic environments. For example, an independently simulated aircraft could serve as a digital twin that mirrors the expected movement of a real aircraft based on clearance and intent. If the simulated aircraft turns left while the actual aircraft continues straight, this divergence could signal a potential deviation or pilot error, warranting an alert to human controllers. Such mechanisms could enhance runway safety and provide real-time validation of aircraft compliance with ground instructions.

Thus, the use of RL agents not only facilitates the autonomous operation of aircraft within the simulation but also lays the groundwork for future systems capable of active monitoring, decision support, and anomaly detection in real-world environments.

4.2 Reinforcement Learning

Reinforcement learning (RL) is a branch of machine learning focused on training agents to make a sequence of decisions by interacting with an environment. In contrast to supervised learning, where the model learns from labelled examples, an RL agent learns from experience where it receives feedback in the form of rewards based on the outcomes of its actions. This feedback is used to refine the agent's policy, i.e., the strategy it uses to select actions in different situations [51].

The RL problem is typically formalised as Markov Decision Process (MDP), defined by the tuple (S, A, T, R, γ) , where:

- S is the set of all possible states the agent can observe (e.g., the aircraft's current position, heading, and distance to the next waypoint),
- A is the set of available actions (e.g., turn left, go forward, slow down),
- $T(s, a, s') = P(s_{t+1} = s' | s_t = s, a_t = a)$ is the transition probability of moving from state s to s' when taking action a
- $R(s, a)$ is the reward function, which maps state-action pairs to scalar feedback values,
- $\gamma \in [0,1]$ is the discount factor, which determines the importance of future rewards.

The objective in reinforcement learning is to learn a policy that maximises the expected sum of future rewards, also known as the return. A central concept in many RL algorithms is the action-value function, or Q-function (i.e., the ‘quality’ of an action), denoted $Q(s, a)$. It represents the expected return of following an optimal policy by taking action a in state s . The relationship that defines $Q(s, a)$ is known as the *Bellman equation*:

$$Q(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(s', a')$$

The Bellman equation essentially says that the quality of a state-action pair equals the immediate reward plus the best reward possible from the next state (which is weighted by the transition probability and discount factor).

In the context of this project, reinforcement learning is particularly well-suited for controlling aircraft on the ground for several reasons. Most importantly, the task is inherently sequential as aircraft must make a series of decisions to complete a route from gate to runway (or vice versa). RL enables agents to learn optimal policies from trial and error within the simulation environment.

4.3 Q-Learning

Q-learning is a reinforcement learning algorithm used to estimate the value of taking a particular action in a given state [51]. It is described as a model-free algorithm, meaning that it does not require any prior knowledge of how the environment works (i.e., no need to model the transition dynamics or reward distribution). It is also off-policy, meaning that it can learn the best action-value function independently of the policy being followed during learning.

The core idea in Q-learning is to learn a Q-function, $Q(s, a)$, which estimates the expected collective reward of taking action a in state s , and then following the optimal policy. As the agent interacts with the environment, it updates this function using the following rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]$$

Where:

- s_t is the current state
- a_t is the action taken
- r_t is the reward received

- s_{t+1} is the new state
- α is the learning rate, controlling how much new information overrides old estimates
- γ is the discount factor, which determines the importance of future rewards

This update rule is applied after the agent takes an action. The term $\max_{a'} Q(s_{t+1}, a')$ represents the estimate of the best future reward from the next state, which it uses to adjust its current estimate for $Q(s_t, a_t)$.

To decide what actions to take, the agent follows an epsilon-greedy strategy meaning that with probability ϵ , it selects a random action to encourage exploration, and with probability $1 - \epsilon$, it chooses the best action (highest Q-value) to exploit current knowledge.

Q-learning is practical in small environments. However, it does not scale well to problems with large or continuous state spaces. In these cases, keeping a table of Q-values for every possible state-action pair becomes infeasible. This limitation makes it unsuitable for more complex tasks such as autonomous aircraft navigation, where the agent must take many variables into consideration. To overcome this limitation, neural networks can be used to approximate the Q-function, allowing for learning in complex environments [52].

4.4 Deep Q-Learning (DQL)

Deep Q-Learning (DQL) extends the classical Q-learning algorithm by using a neural network to approximate the Q-function instead of storing values in a table. In the context of this application where each aircraft will process continuous information such as position, heading and speed, a function approximator like a neural network is essential [52, 53].

In DQL, a deep neural network takes a state s as input and outputs estimated Q-values for all possible actions a . This network is referred to as the Q-network, and its parameters θ are updated during training to minimise the difference between predicted and target Q-values.

The network is trained by minimising the following loss function:

$$\mathcal{L}(\theta) = \left(r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-) - Q(s_t, a_t; \theta) \right)^2$$

Here:

- $Q(s_t, a_t; \theta)$ is the predicted Q-value for the current state-action pair,
- θ^- refers to the parameters of a separate target network, which is updated less frequently to stabilise learning,
- γ is the discount factor,
- r_t is the immediate reward.

The use of a separate target network helps prevent extreme updates during training. By updating the target network at sparse intervals, the algorithm avoids chasing a moving target when computing the expected future reward. During training, the target Q-value is computed using the target network, while the loss is propagated through the main Q-network only [52].

Another important feature of DQL is *experience replay*. Instead of learning from only the most recent transition, the agent stores its experiences (s_t, a_t, r_t, s_{t+1}) in a replay buffer. During training, it samples random batches from it. This gives the benefits of breaking the correlation between sequential observations and allowing an agent to learn from past experiences multiple times.

Due to the architecture of Deep Q-Learning, it can scale to problems that are much more complex than those solvable with vanilla Q-learning. In this project, DQL was used to train an aircraft to make movement decisions such as turning and accelerations while navigating the taxiway network. In the next section, the specifics of this implementation are described in detail.

4.5 Implementation

4.5.1 Training Ground Setup

To train a model capable of navigating a taxiway network, a dedicated training environment was developed separately from the main airport simulation. This was done to promote generalisation across varied taxiway shapes and layouts, and to prevent overfitting to the fixed, grid-like structure of Birmingham Airport. In this setup, each training episode generated a random route comprising three waypoints. The agent was spawned at the start of the route, initially facing toward the first waypoint. When the agent got within a certain distance of the goal waypoint, the state would update to inform the agent of the next waypoint's location.

The `generate_path()` function produced the three waypoints sequentially. Each new point was sampled randomly within the display but was only accepted if its distance from the previous point fell between 50 and 500 pixels. This ensured that route segments were neither too short nor too long. A further constraint was added to ensure that the angle between consecutive segments did not exceed 90 degrees. This was necessary because, without it, the environment occasionally produced unrealistic, sharp turns that would be infeasible for an aircraft to follow. Figure 4.A illustrates the difference in route generation before and after this constraint was applied.

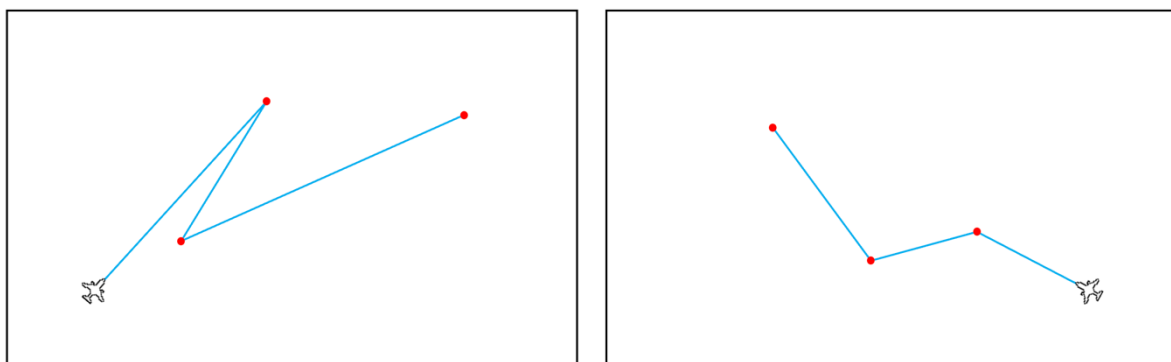


Figure 4.A: Route generation with and without angle constraint. Left: no constraint applied. Right: 90° angle constraint enforced.

4.5.2 Initial Implementation and Challenges

A separate training environment was developed to simulate an episode loop. Each episode would end if the aircraft deviated too far from the route, went out of bounds, or successfully completed the path. A maximum frame count of 1000 frames were enforced that was long enough to complete the route threefold ideally. The script saved model checkpoints at regular intervals and whenever a new high score was achieved. These checkpoints could then be evaluated in a local, non-training environment to visualise learning progress.

The initial model was a standard Deep Q-Network (DQN), with an input layer matching the dimensionality of the observed state space. Experiments were conducted with one and two hidden layers, using sizes ranging from 16 to 128 neurons. Each episode averaged around 2,000 frames, so the replay buffer was set to store 500,000 transitions (about 250 episodes). A batch size of 5,000 samples was used during training. The action space was made up of five actions: turn left, turn right, increase thrust, decrease thrust, and do nothing.

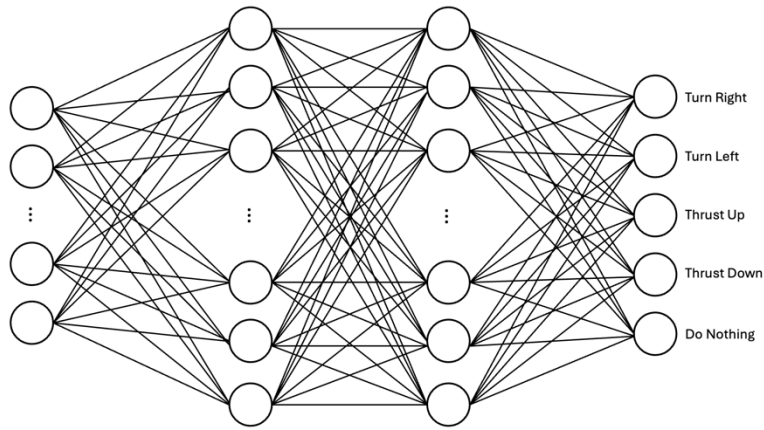


Figure 4.B: Network architecture used in initial DQN model.

The first training objective was to teach the aircraft to maintain “ideal” movement in relation to the centreline of the taxiway. Based on real-world taxiing norms, aircraft should move at approximately 15 knots during turns and 25 knots on straights [54]. This was modelled using an “ideal region” that moved along the route at the expected speed. The aircraft was rewarded for staying within this region. A visual example of this region visualised as a yellow circle can be seen in Figure 4.C.

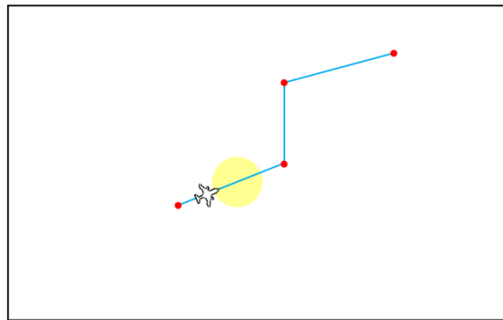


Figure 4.C: Visualisation of the ideal movement zone (yellow), which the agent was incentivised to stay within.

Originally, the state input included the aircraft's x and y coordinates, heading, thrust setting, speed, the goal waypoint's x and y coordinates, the angle to the goal, and the distance from the centreline. All values were normalised between 0 and 1.

The reward function was constructed with several components:

- A small positive reward for every 3 pixels of progress toward the next waypoint
- A smaller reward for remaining within the ideal region
- A penalty for exceeding speed or deviation thresholds
- A large reward for successfully reaching a waypoint

Despite extensive tuning of reward weights, batch sizes, and network configurations, training remained very noisy and inconsistent. The best observed policy exploited the reward structure by accelerating rapidly in a straight line toward the first waypoint (earning rewards for progress and arrival) then exiting the bounds and receiving a penalty. This behaviour displayed an inability to learn how to perform turns.

To address this, the training environment was modified to include a small random variation (up to 15 degrees) in the aircraft's heading when spawned. While this change encouraged the agent to learn to turn in the correct direction and accelerate toward the goal, it consistently failed to understand when to stop turning.

4.5.3 Input Representation and Reward Redesign

After exhausting changes to the previous implementation, it became clear that the main issue was likely in the structure of the input state. The agent lacked sufficient spatial awareness of its relationship to the goal, especially when it came to turning and alignment. A complete redesign of both the state representation and reward function was then attempted.

An alternative approach was first tested that laid the groundwork for the final working solution. This method used a directional "cone of vision" model. The aircraft's surrounding space was divided into eight sectors, and the cone containing the goal waypoint was encoded as a one-hot vector (one if the goal was in that direction, otherwise zero). This representation provided the agent with some directional awareness without relying on absolute coordinates. In retrospect, this method would also have scaled better to airports of varying sizes, where raw x and y coordinates might not generalise effectively. In addition to this, the aircraft's heading, speed, and thrust setting were included in the input state vector.

While this implementation did not result in successful training, it served as an importation stepping stone. It demonstrated that abstracting spatial relationships into structured, discrete signals could be more effective than supplying raw positional data. Alongside this, the reward function was redesigned. The previously used "ideal zone" was removed, as the agent lacked any information on it. Instead, rewards were given for turning in the correct direction (e.g., if the goal was to the left, the agent was rewarded for turning left) and for maintaining an acceptable speed range.

However, when tested, the agent often became stuck turning indefinitely in one direction. This was likely because at longer distances, the waypoint consistently activated only one cone,

giving the illusion of progress even when the aircraft was not advancing. While this could have been avoided by using smaller sectors, this would have significantly increased the state space.

To address this, a second approach was introduced, inspired by ray tracing. Instead of using abstract directional cones, the agent received sensor-like input that simulated “beams” projecting from the front, left, and right sides of the aircraft. These beams returned the distances to the nearest taxiway edges in their respective directions. This greatly reduced the state space while still conveying the aircraft’s spatial constraints. Combined with the relative angle to the goal, the distance to the goal, speed, and thrust setting, this formed the final input representation.

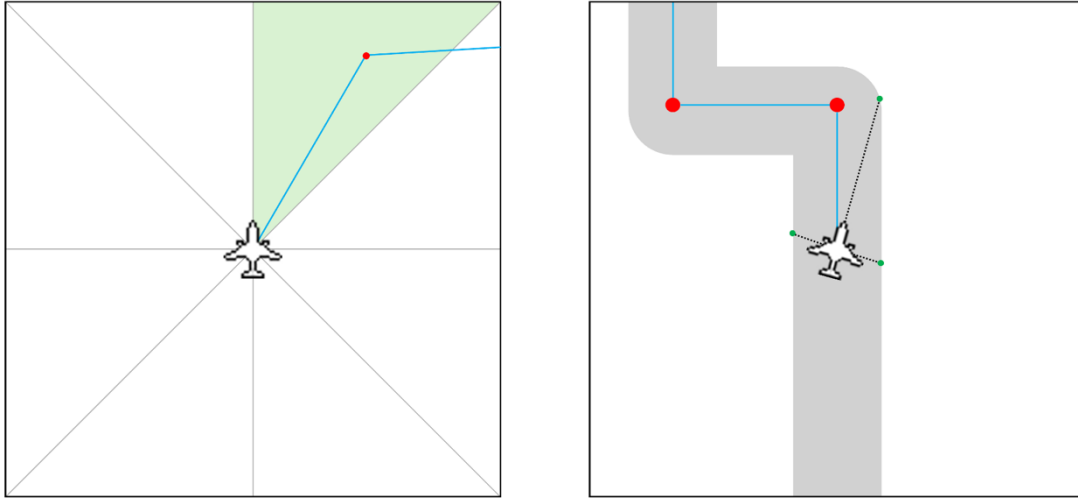


Figure 4.D: Comparison of input space representations. Left: Cone-based directional encoding using eight one-hot sectors to locate the waypoint. Right: Ray tracing-inspired model using beams to measure distance from taxiway edges on the left, centre, and right.

While this configuration was more effective, the agent began to exploit the reward function. It would remain stationary but continually rotate toward the correct heading, thereby receiving positive rewards for turning correctly without making progress. The issue was due to how the rewards were being backpropagated through the network. Because the reward was shared across all output actions, the agent could not differentiate whether it was being rewarded for correct turning or penalised for lack of movement. This insight gave the motivation for a change in neural network architecture that would help direct rewards better.

4.5.4 Multi-Headed Deep Q-Network Architecture

The main issue identified in the previous section was that the standard DQN backpropagated the loss across the entire output layer, regardless of which type of action the agent was executing. This meant the model struggled to differentiate which decisions were contributing to reward and which were not. In the previous setup, the agent selects a single action per frame (the one with the highest Q-value when not exploring). This choice is either a steering action (turn left/right), a speed control action (thrust up/down), or doing nothing.

What is needed for this learning task is to simultaneously evaluate both output types and take both actions, thus, what is required is a mechanism to learn and understand these controls independently. This is where Multi-Headed Deep Q-Networks (MHDQNs) were introduced.

An MHDQN extends the standard DQN architecture by splitting the output layer into multiple independent “heads,” with each head responsible for a distinct subset of actions [55]. In this case, one head governs steering actions, and the other controls thrust. During training, rewards are computed and backpropagated separately for each head, allowing the network to learn appropriate Q-values for different action types without them interfering with each other. During inference, the outputs of both heads are used together to perform a compounded action.

The model was subsequently changed to a MHDQN, with two output heads implemented. The first head controls steering, with actions set to turn left, turn right, and no turn. The second head controls thrust, with actions being increase thrust, decrease thrust, and maintain current thrust. This architectural change led to noticeably more stable training and significantly improved results from the previous architecture. A visual representation of the MHDQN architecture used in this project is shown in Figure 4.E.

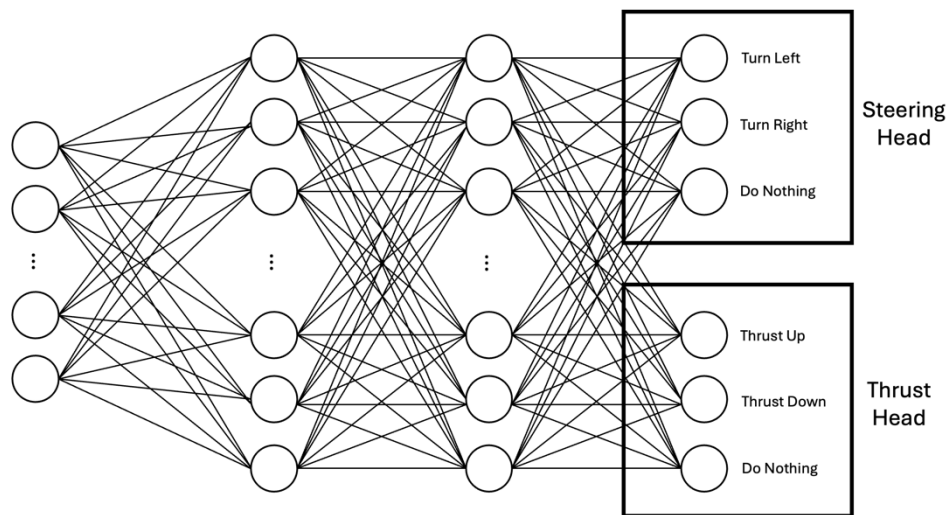


Figure 4.E: Architecture of the MHDQN used for training.

4.5.5 Final Implementation and Successful Training

The final reward function was structured to provide clear, directed feedback for each of the two output heads. For steering, the agent was rewarded or penalised based on how accurately it responded to the angle between its current heading and the target direction:

- +0.5 for turning in the correct direction (left or right) when misaligned
- -1.0 for turning in the wrong direction
- +0.1 for maintaining a stable heading when no correction was needed

For thrust, the agent was guided toward maintaining a realistic taxiing speed, with rewards shaped around safe operational ranges:

- +0.5 for maintaining a speed between 15 and 20 knots
- -0.2 for speeds falling outside this optimal range

These rewards were propagated separately from each of their respective heads, so the final layer of fully connected nodes was the one specifically making the evaluation of which is the ideal action to take within each head. The localisation of reward signals to their corresponding

output heads addressed the earlier issues by avoiding the ambiguous feedback and lead to more stable learning.

The network architecture used in the final MHDQN implementation was designed to balance simplicity with sufficient capacity for learning motion behaviours. The input layer accepted a state vector composed of:

- Ray-traced distances to the left, front, and right of the aircraft
- Relative angle to the waypoint
- Distance to the waypoint
- Aircraft speed
- Thrust setting

This totalled 7 input features, all normalised between 0 and 1. The shared base of the network consisted of two fully connected hidden layers, each with 16 nodes and ReLU activation functions. From the shared base, the network split into two separate output heads:

- The steering head produced three Q-values representing the actions turn left, turn right, and no turn.
- The thrust head produced three Q-values for increase thrust, decrease thrust, and maintain thrust.

These hyperparameters for the final model were chosen through educated observations across multiple training runs and formed the basis of the final model that achieved stable and correct taxiing behaviour. The following hyperparameters were used in the final training configuration (it is important to note that the hyperparameters remained very consistent throughout training all approaches):

- Discount factor ($\gamma = 0.99$): This high value creates strong emphasis on the long-term rewards, hence encouraging the agent to make decisions that lead to the successful navigation of the whole route rather than short-term gains.
- Learning rate: 0.001: The network was optimised using the Adam optimiser [43] with a fixed learning rate, providing a good balance between convergence speed and stability during weight updates.
- Exploration strategy: An ϵ -greedy policy was applied to balance exploration and exploitation during training. The value of ϵ decayed exponentially using the rule

$$\epsilon = \max(0.05, 1.0 \times 0.995^n),$$

where n is the number of episodes. This ensured the agent explored sufficiently in early training while focusing more on exploitation as performance improved but was capped to a minimum of 5% chance of exploring rather than exploiting.

4.6 Evaluation and Results

The evaluation of the final model was conducted primarily through direct visual observation within the simulation environment. The aim was to assess whether the agent could reliably

navigate taxiways, maintain acceptable speeds, and generalise across a variety of configurations. The entire iterative development and training process spanned over 200,000 episodes across just over a week. Once the final model design was established, a focused training run over 4,000 episodes (completed in approximately four hours) produced the final policy.

A graph of training performance is shown in Figure 4.F, illustrating the plateau in reward after about 2,000 episodes that indicated convergence and policy stability.



Figure 4.F: Training curve for the final MHDQN model

The final model demonstrated the following behaviours consistently:

- Maintained a lateral deviation of no more than 3 pixels from the centreline (7 meters)
- Operated at an average taxiing speed of 16 knots (within the desired 15–20 knot range)
- Reached goal waypoints without erratic movement or oscillations

One validation test involved manually setting the ray-traced distance inputs to zero, simulating a case where the aircraft was not on the correct taxiway. In this situation, the agent correctly refused to move, indicating that it was not overly reliant on a single feature (such as the angle to the goal), and had learned to reason holistically about its environment. The trained model was successfully deployed across multiple aircraft simultaneously, with each instance running independently and maintaining stable motion throughout the simulation.

A limitation that was identified accidentally during later phases of development was that on very long, straight taxiways, the aircraft tended to drift significantly to the right over time. This issue was not observed during training due to a cap on the maximum taxiway length in the training environment. The problem was resolved by inserting intermediate waypoints along extended routes, allowing the agent to periodically re-align its heading.

5 Aircraft Coordination and Conflict Management

5.1 Overview

What has been implemented so far is a simulation that allows individual aircraft to navigate between different points on the taxiway network of an airport. Each aircraft follows its own route towards a specified goal. At this stage, aircraft have no information on the presence of other aircraft in the airport. There is no collision logic, no concept of right-of-way, and no decision-making related to shared infrastructure such as taxiway intersections, holding points, or runway entries. As a result, the behaviours witnessed are unacceptable in a real-world system, such as planes passing through one another or blocking each other's paths.

To address this, an intelligent mechanism must be introduced, one that can coordinate and manage movement conflicts, enforce priorities, and allow agents to yield or stop when appropriate. In ground control operated, these functions are handled by human air traffic controllers who issue clearances based on situational awareness and scheduling constraints. In the system developed here, the aim was to replicate this logic through decentralised behaviour in a way that each aircraft can decide when to make way for others.

A logical approach for this kind of problem, where multiple independent agents must share an environment, is Multi-Agent Reinforcement Learning (MARL) [56], specifically using a Centralised Training with Decentralised Execution (CTDE) framework, inspired by ongoing research such as Ghosh et al. [31]. This framework has agents train together using information shared globally but operate independently during inference. This approach appeared to be the best direction for decentralised coordination without explicit communication mechanisms between aircraft.

5.2 Initial MARL Approach (CTDE)

5.2.1 Setup and Architecture

MARL builds on many of the core ideas introduced in the last chapter, but with the added complexity of handling multiple agents. In this setup, there is more flexibility, like the potential for communication between agents. Some MARL implementations use attention mechanisms as a form of message passing between agents. However, in this case, a simpler and more efficient approach was employed using a Centralised Training with Decentralised Execution (CTDE) framework.

Under CTDE, all aircraft agents shared a common Q-network, and their experiences were stored in a single shared replay buffer [57]. This setup allowed the agents to train on the collective experiences of all aircraft, but there was no explicit communication between them. Instead, the communication was implicit in the sense that shared experiences and policy enabled coordination and cooperative behaviours to emerge. Like the previous reinforcement learning model, a separate Python training environment was created to ensure a clear separation of the training and operational environments. During training, experiences from all agents were

sampled together and used to update the shared network. When running inference, each agent acted independently, using the same trained model to make decisions in real-time, mimicking the decision-making process of an air traffic controller who would issue instructions to each aircraft.

The CTDE framework has the advantage of being able to account for variability in the number of agents. Since the airport environment is dynamic and the number of aircraft can change throughout, this centralised training approach can process information from different numbers of agents, allowing the model to learn generalisable cooperation strategies. Each agent makes independent decisions based on its local observations, eliminating the need for constant communication between agents, which would be computationally expensive as the number of agents increases.

5.2.2 Q-Network Structure

The challenge of modelling coordination among multiple aircraft is that each agent must learn how to react to its local surroundings. Inferring from training in the last chapter, it was decided that the best input would be a small, interpretable observation of the agent's local environment. The goal was for each agent to understand its own position and the position of its nearest neighbours within a 100-pixel radius. This seemed like the most logical approach because the relevance of other planes depends on their proximity and trajectory. For example, an aircraft entering the runway does not need to know about a plane on the other side of the airport. By focusing on the nearest two aircraft, this approach allowed the agents to make contextually informed decisions about yielding and moving.

The input space for each agent consisted of:

- Aircraft speed
- Distance to the next waypoint
- Distance to the final waypoint

For each of the two nearest aircraft:

- Speed
- Distance to the aircraft
- Relative angle to the aircraft
- Aircraft's distance to its final waypoint

This resulted in roughly 11 input features, which were normalised between 0 and 1. Some experimentation was done using sine and cosine functions for the headings to ensure smooth continuity of heading values, but the core inputs remained throughout. The reasoning was that heading information is implicitly encoded in the angles between agents, so explicit heading data was not needed. The model's output consisted of two action decisions: Stop or Go. These were binary actions, with each output node corresponding to one of the actions. The action with the maximum Q-value was selected during training and inference. Based on the results from the previous RL model, two hidden layers were used, and their sizes were experimented with between 32 and 256 nodes.

5.2.3 Reward Function and Coordination Logic

The initial reward function was designed to incentivise a correct yielding behaviour. The logic was based on the relative positions of nearby aircraft. The two nearest neighbours to each plane were identified, and their relative headings and positions were calculated. If a plane was within -45 to $+45$ degrees of the agent's heading and within 40 pixels (100 meters), it was considered in the agent's "cone of vision".

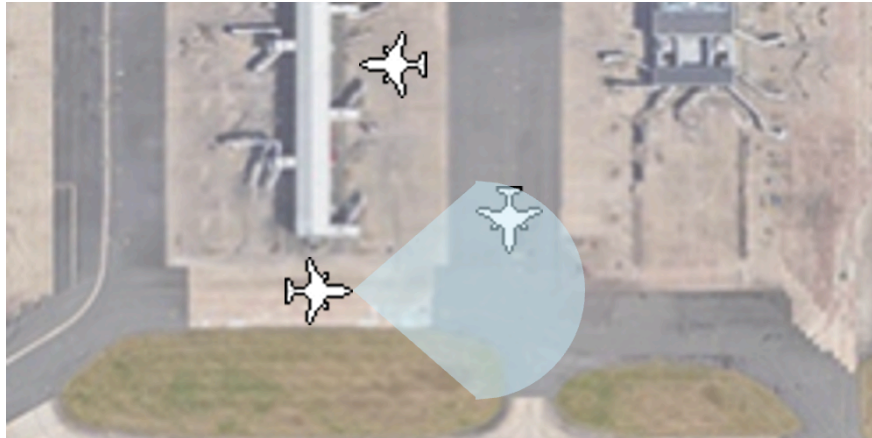


Figure 5.A: A scenario where an agent had another aircraft in its cone of vision and the ideal action is to yield

If the plane had a higher priority (i.e., closer to its destination), the agent should yield:

- Reward for stopping: +3
- Penalty for continuing to move: -5

If the agent had priority over the other plane:

- Penalty for stopping unnecessarily: -2
- Reward for continuing to move: +2

If there were no planes in the "cone of vision":

- Penalty for stopping unnecessarily: -1
- Small reward for continuing: +0.1

The reward function was designed to mimic the decision-making logic of ATC. When a plane needed to yield, it was rewarded for doing so, and when it was allowed to proceed, it was incentivised to move forward. However, this reward function was not fixed throughout training. Adjustments were made based on empirical observations that agents were not consistently learning how to stop at the correct times, but the general scale of rewards and penalties remained constant throughout the experimentation of different reward quantities. This led to the hypothesis that the reward structure was too rigid and lacked flexibility.

5.2.4 Initial Approach Failure

Multiple training runs were conducted with this setup, using the shared replay buffer and the same network for all agents. Throughout the training process, the primary changes involved adjusting the number and sizes of hidden layers and experimenting with reward weights. Hyperparameters such as the discount factor and learning rate remained the same as the ones in the previous aircraft movement model (0.99 and 0.001 respectively). The goal was to have agents to be able to understand their relative position to nearby aircraft and make independent yet coordinated decisions during inference.



Figure 5.B: Unstable and noisy training curve in the initial MARL implementation

Unfortunately, training proved to be extremely noisy, with little visible progress. As shown in Figure 5.B, the training oscillated and was highly unstable. After trying different inputs and reward modifications, the conclusion was that the reward function itself was too rigid and lacked the flexibility needed to capture the dynamic nature of coordination. The agents struggled to learn when to stop or continue moving, as the rigid reward structure likely misclassified many situations. This prompted the need for a more dynamic and context-aware reward function, and the introduction of trajectory prediction models became the next step in improving coordination.

5.3 Introducing LSTM-Based Context Modelling

5.3.1 Motivation and Approach

Since hardcoding a reward structure proved to be challenging and inaccurate, a refined and context-aware reward system was needed to overcome this issue. The idea was to predict the future locations of aircraft and determine whether they were likely to conflict. If the predicted locations of two aircraft overlapped at the same future timestep, it would be interpreted as a potential collision. Using this mechanism, coordination could then be learned by rewarding aircraft for yielding appropriately before the conflict occurred.

Recurrent neural networks (RNNs) were the natural candidate for temporal prediction, and particularly the Long Short-Term Memory (LSTM) architecture [58]. It was chosen due to its prominently exceptional performance in learning from sequential data and making accurate predictions of future sequences.

The LSTM architecture is designed to model sequences with long-term dependencies. They use memory cells and gating mechanisms to retain relevant information over time. At each timestep t , the hidden state h_t and cell state c_t are updated using input x_t and the previous states. The key operation is: $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$ where f_t , i_t , and $\tilde{c}_t$ are the forget gate, input gate, and candidate cell state (a proposed update to the cell's memory, based on current input and past hidden state), and $\odot$ denotes element-wise multiplication. This structure enables the model to learn which information to keep, update, or forget [58]. This next part outlines how the LSTM model was trained to forecast future aircraft positions, and how these predictions were integrated into the reward mechanism in MARL for more accurate modelling of context.

5.3.2 Dataset Construction and Preprocessing

LSTMs are typically trained on temporal datasets where time relationships are embedded in the sequence of observations. As there are no open-source datasets to model aircraft movements from, a custom dataset was collected from the existing simulation. The goal was to train the model to predict the position and heading of aircraft at future time intervals (for example, at 1, 3, 5, 7, and 10 seconds after a given observation window). To create this dataset, aircraft states were sampled every 0.5 seconds over a 5-second window. Each observation sequence consisted of 10 timesteps (5 seconds), and each timestep included 11 features (all normalised between 0 and 1), which are listed below:

- x-coordinate
- y-coordinate
- Heading
- Speed
- *Encoded State (encoding of the aircraft's operational state)*
- Distance to Next Waypoint
- Heading (Sine)
- Heading (Cosine)

The one-hot encoded² operational states of the aircraft were categorised as follows:

- preparing and parked: [1, 0, 0, 0] – Stationary
- taxiing and parking: [0, 1, 0, 0] – Taxiing
- landing and rollout: [0, 0, 1, 0] – Landing phase
- pushback and takeoff: [0, 0, 0, 1] – Active movement

² One-hot encoding is a method of representing categorical variables as binary vectors, where each class is assigned a unique vector with a single high (1) value and the rest as low (0).

Each row in the dataset contained a sequence of 10 input states and its corresponding future x , y , and heading values at 1s, 3s, 5s, 7s, and 10s after the end of the sequence. A data collection script was run in the background of the aircraft movement simulation to log these values. The simulation ran for approximately eight hours, collecting data from around 3,000 aircraft throughout 200 simulation episodes. In each episode, 20 aircraft (10 for take-off and 10 for landing) were simulated, and the runway in use was alternated between each episode to ensure the dataset was diverse. In total, 2.26 GB of data was collected, amounting to just over 1.2 million datapoints, which were then exported to CSV format for training.

5.3.3 Model Architecture and Training

The LSTM model was implemented using PyTorch. The architecture consisted of:

- Two stacked LSTM layers with 128 hidden units each,
- A fully connected output layer that predicted three values: x , y , and heading.

Since the network was made to predict multiple future steps (3, 5 and 7 seconds into the future), the forward function returned a prediction for each timestep. This meant that each prediction made by the model would be fed back in as input to calculate the next timestep. In code:

```
def forward(self, x):
    lstm_out, _ = self.lstm(x)          # (batch, seq_len, hidden_size)
    out = self.fc(lstm_out[:, -1, :])    # Predict from last timestep
    out = out.view(-1, 3, 3)             # Output for 3 timesteps, 3 values each
    return out
```

Algorithm 2: Forward pass of the LSTM network to produce multi-step predictions.

The model was trained using a supervised learning loop with early stopping³ implemented based on validation loss. The dataset was split 80/20 into training and validation sets. Training was stopped when validation loss failed to improve for five consecutive epochs.

- Training completed after 38 epochs.
- Initial train/val loss: 0.0033 / 0.0020.
- Final train/val loss: 0.0010 / 0.0009.
- The average prediction error was within 10 pixels total for x and y , and at most 35 degrees for heading.

While the model's accuracy was extremely good and reliable, to be safe and avoid extreme deviations, the predicted positions were snapped to the nearest point on an aircraft's planned route, effectively constraining the predictions to the taxiway centreline when on the ground.

5.3.4 Integration Into Reward Logic

To test this implementation's usefulness for coordination, the LSTM model was used to simulate future positions for all aircraft in each frame. When two predicted positions overlapped at the same timestep, the aircraft closer to its destination was given priority, while

³ Early stopping stops training when validation loss stops improving after a number of epochs to avoid overfitting.

the other was forced to yield. This implementation produced the desired coordination behaviour where aircraft yielded when appropriate, and conflicts were resolved naturally. Since the LSTM-based logic worked under a hard-coded implementation, it was integrated into the MARL training loop to provide a dynamic and more contextually accurate reward function.

Despite the more informative and accurate reward function, the MARL model still failed to learn consistent coordination. What was observed was a highly liberal policy, meaning that aircraft rarely chose to stop. During training, they often learned that the fastest way to receive a reward was to continue moving and reach the runway, regardless of other agents. The likely cause of this failure was the training implementation itself, specifically in early parts of an episode, when all aircraft were still at their stands, there was minimal congestion and no need to yield, and episodes of restored at the first instance of a collision. After many episodes of this occurring, ϵ -greedy exploration had already decayed, which made agents default to exploiting the behaviour of continuing to move forward and never stopping.

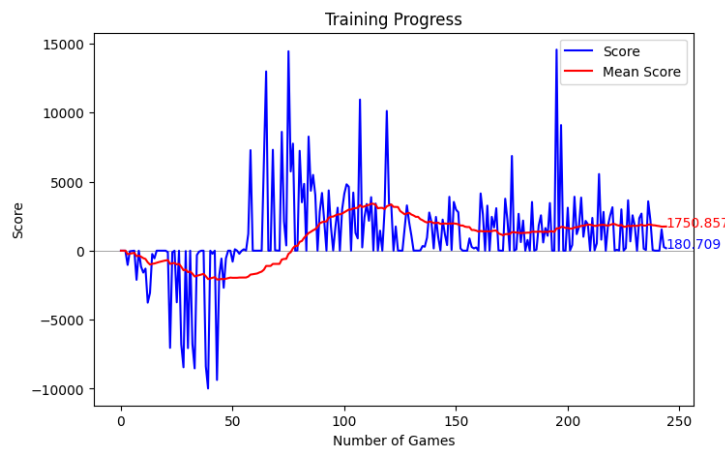


Figure 5.C: Training scores over LSTM-based reward implementation.

As seen in Figure 5.C, the initial episodes in this implementation show erratic performance due to frequent collisions and variable episode lengths. As training runs progressed the policy plateaued with high variance, reflecting the overreliance on forward movement and minimal yielding. This prompted further investigation into alternative approaches, which are discussed in the next section.

5.4 Reward Shaping and Experience Replay Techniques

5.4.1 Skewed Reward Distribution and Replay Buffer Analysis

To verify the theory that scenarios involving stopping were sparse in the replay buffer, a distribution analysis was conducted where the expected action frequencies were compared to actual occurrences stored during training. The findings were as follows:

- Expected ratio of *Go* to *stop*: 91% to 9%
- Of the *Go* actions:
 - 80% were correctly executed
 - 20% were incorrect
- Of the *Stop* actions:

- 6% were correctly executed
- 94% were incorrect

This showed a strong skew in the number of correct stopping instances available to train from. The model was exposed to *Go* transitions in an overwhelming manner, and its policy reflected this bias. Before implementing any specific strategies to fix this skew, the simulation logic was modified such that when a collision occurred, only the aircraft involved in the collision were despawned, while others continued. This gave the remaining agents more opportunities to explore, rather than resetting the entire episode prematurely.

5.4.2 Balanced Experience Replay (BER)

Balanced Experience Replay (BER) was first implemented to attempt correcting the imbalance in the replay buffer. In normal experience replay, transitions are stored in a single buffer without regard for their type or significance. As a result, common scenarios (like “go” decisions) dominate the training samples, while more rare cases (such as yielding decisions) are underrepresented. BER fixes this by dividing transitions into separate buffers based on the action taken and the correctness of that action [59]. For this implementation, four buffers were implemented for each of the possible actions and their correctness:

- Good Stop – stopping when expected
- Bad Stop – stopping when not required
- Good Go – moving forward when clear
- Bad Go – failing to yield

Each buffer was allocated a maximum of 100,000 transitions, and samples were drawn evenly from each. Formally, if the experience replay size was N , then $\frac{N}{4}$ transitions were sampled from each of the four categories. If a buffer contained fewer than the required number of samples (which was tested to vary between 50 and 500), oversampling⁴ was used to pad the batch. During training, the captured transitions were classified based on the action and reward and inserted into their appropriate buffer immediately after an action was taken. Transitions were classified based on the action taken and the resulting reward. For example, a *Stop* action with a positive reward was a *Good Stop*, while a negative reward indicated a *Bad Stop*; the same applied to *Go* actions.

Unfortunately, the results of BER were consistently poor. In the early stages of a training run, the agent’s actions were constantly random, and as training progressed, the learned policy seemed to reverse the behaviours expected. In most training runs, the agent became very conservative where it refused to move in situations where it had priority.

⁴ Balancing an imbalanced dataset by duplicating samples from the minority class.

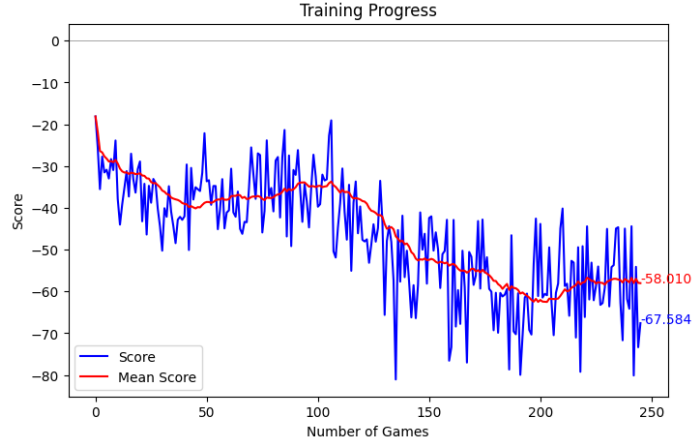


Figure 5.D: Training performance under Balanced Experience Replay.

This consistent decline in performance led to the assumption that BER had over-corrected for rare events, and that introducing a high frequency of rare cases had the model overfit to it. As shown in Figure 5.D, training performance continued to deteriorate and showed no indication of the agent learning a useful policy.

5.4.3 Prioritised Experience Replay (PER)

To fix the issue of BER where it balances scenario types explicitly, Prioritised Experience Replay (PER) selects experiences based on their learning potential, instead of sampling uniformly from the replay buffer. The goal is to train the model more efficiently by focusing on transitions that the model has not yet been exposed to enough to learn, and those can be referred to as the ones that are more “surprising” or informative [60]. PER achieves this by assigning each transition a priority score based on its temporal-difference (TD) error, defined:

$$\delta = \left| r + \gamma \max_{a'} Q_{\text{target}}(s', a') - Q_{\text{online}}(s, a) \right|$$

where r is the reward for the transition, γ is the discount factor and Q_{target} and Q_{online} are the target and online Q-networks, respectively. Transitions that have larger TD errors mean they occur less frequently and represent more niche situations, so they are prioritised during sampling. Each transition is sampled based on a probability:

$$P(i) = \frac{p_i^\alpha}{\sum_j p_j^\alpha}$$

where p_i is the priority of transition i , p_j is sum of all priorities and $\alpha \in [0,1]$ is a hyperparameter which determines how much prioritisation is applied (higher means more bias towards high-priority samples). Since this type of sampling creates bias, PER corrects this by using importance sampling weights defined as:

$$w_i = \left(\frac{1}{N \cdot P(i)} \right)^\beta$$

where N is the total number of transitions in the buffer, $P(i)$ is the sum of probabilities and $\beta \in [0.4, 1]$ is a hyperparameter that controls how aggressively this correction is applied over time. The loss is then scaled by w_i to ensure updates are fair across samples and that no skew is introduced. This mechanism encourages the agent to focus more on the rare cases, like correct stopping behaviours, which were underrepresented in earlier training runs.

Despite the theoretical advantages of PER, it came with a large computational cost. Because priority scores and probabilities are updated every frame, training was observed to be over 10 times slower compared to standard experience replay and BER. One training run could take up to 6 hours (~ 250 episodes) before seeing any meaningful results to decide the success of that training run. However, early training results were the most promising seen up to this point.

Figure 5.E shows a steady rise in performance, suggesting that PER was helping the agent learn some effective policy, but plateaued initially. With some tuning of the reward sizes and PER hyperparameters, the models began to increase above a neutral score. This is shown in Figure 5.F, where the learning curve rises well above zero and exhibits a strong upward trend.

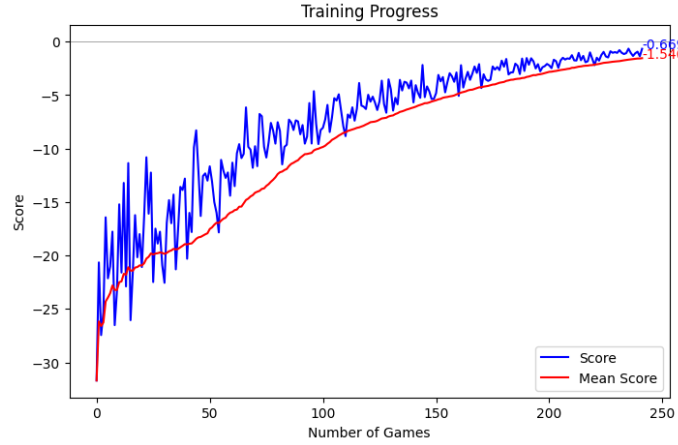


Figure 5.E: Early performance of PER training.

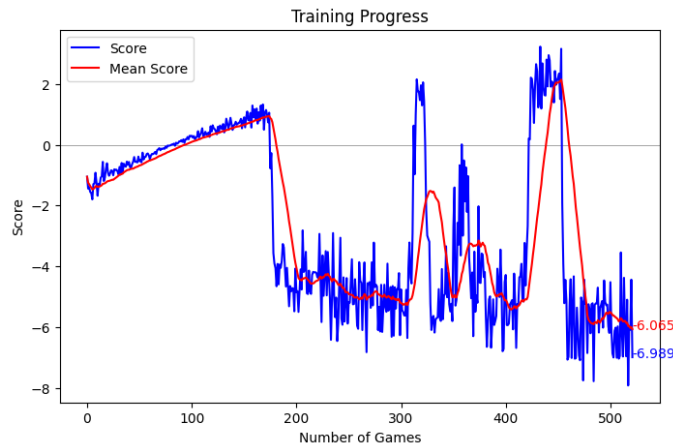


Figure 5.F: PER with reward and hyperparameter tuning.

Unfortunately, for reasons that remain unclear, given enough training time, catastrophic crashes occurred mid-training with PER. The scores would peak and then collapse, sometimes recovering briefly before collapsing again, and this instability became a common feature across multiple training runs. At the peak, the agent demonstrated a somewhat reasonable

learning of the policy, occasionally choosing to stop and yield appropriately. While the behaviour was not perfect, it was clearly better than random behaviour. After the crashes occurred, the model always defaulted back to an overly liberal policy meaning that it rarely ever stopped. This recreated the exact same behaviours seen in earlier failures.

By looking at the training curves, two main causes were hypothesised: exploding gradients⁵ and catastrophic forgetting⁶. Common fixes to these issues were attempted, such as gradient clipping and increasing β over time to strengthen the importance sampling corrections. Ultimately, none of these changes resolved the issue, and the best models produced by PER proved to be unstable and unusable for the final simulation.

5.4.4 Other Approaches

When none of the solution implementations produced usable results, the focus shifted toward analysing how the simulation itself was structured and whether its properties were accidentally negatively impacting learning.

Frame Skipping

During external research, it was found that most successful implementations of RL in games or multi-agent environments involved way fewer decision points per episode, less than 1,000 decisions per episode [61]. In this MARL setup, a single episode could span up to 10,000 frames, with the agents making a decision at every frame. This likely contributed to noisy, redundant learning signals and an unnecessarily large replay buffer. To address this, a frame skipping strategy was introduced such that instead of deciding at every single frame, each action would be persisted across 10 frames, and the rewards accumulated during those frames would be averaged and returned at the end of the skip interval. For example, if the agent chose to stop, that same stop command would be held for 10 frames, and reducing the number of decisions to a maximum of 1,000 per episode. However, this approach did not give any new results. The agent became overly conservative like those observed in earlier failed attempts.

Smaller Replay Buffers

Following on from frame skipping, another hypothesis was that a smaller replay buffer might benefit training by keeping experiences more relevant and recent [62]. Since frame skipping slows down how quickly the buffer is filled (by a factor of 10), the buffer size was reduced to as low as 25,000 transitions. Despite this change, the same behaviours persisted, and the agent still failed to learn meaningful coordination.

Global Reward Assignment

Another experimental idea was to treat the system as a single population of aircraft sharing one collective goal instead of separate independent agents. In this context, a global reward was calculated based on the total success and failure of all agents during a single decision step, then this global reward was assigned to all agents, regardless of whether they individually took the

⁵ Gradients become very large during training and cause unstable updates and diverging the weights.

⁶ Models forget previously learned knowledge after learning new information, often due to overwriting of weights.

correct action. In practice, this made learning worse. Assigning the same reward to all actions taken in a frame (whether correct or not) resulted in extremely noisy training curves and the model failed to associate correct decisions with positive rewards. The outcome was a liberal, never-stop policy, where agents ignored all conflicts and moved forward nonstop.

None of these experimental implementations led to the emergence of any cooperative or intelligent behaviour. After exhausting these strategies, it became clear that the simulation structure, reward feedback, and learning objectives were not aligned in a way that enabled MARL to succeed in this setting. As a result, the focus of this DL component was shifted to a non-learning-based solution, which is described in the next section.

5.5 Supervised Learning for Binary Decision-Making

Despite exhaustive experimentation with MARL, there was no clear reason as to why it failed. All modifications done that should have worked either failed or led to unstable policies. This raised the possibility that the approach itself may have overcomplicated a relatively simple task. Given the binary nature of the decision (stop or go) it was decided to try a standard ML classification approach.

5.5.1 Dataset and Feature Engineering

Using the LSTM-based prediction system as ground truth, a dataset was generated where each sample was labelled based on whether the aircraft should stop (1) or go (0), according to future predicted collisions. The input features were the same as those used in MARL, initially including both the observing aircraft's own state and the state of its two nearest neighbours.

- A total of 120,000 datapoints were collected
- The dataset was balanced 50:50 between stop and go actions
- The dataset was split 80/20 into training and validation sets
- Both min-max normalisation and standardisation⁷ were tested, with standardisation performing slightly better

Three ML classifiers were tested:

- **Logistic Regression (LR):** Models the probability of a binary outcome using a linear combination of inputs passed through a sigmoid function.
- **Decision Tree (DT):** Splits data into branches based on feature values, forming a tree of conditional logic.
- **Random Forest (RF):** Builds a group of decision trees on random data subsets and clusters their predictions for robustness.

These models were chosen for their simplicity, ease of implementation, and interpretability, while offering good performance on complex data [63]. This makes them well-suited for initial an evaluation and feature analysis.

⁷ A data transformation that scales features to have a mean of 0 and a standard deviation of 1.

Initial results showed 90%+ accuracy across all models. However, an analysis of the feature importance revealed that the models were overly reliant on the aircraft's own speed. This was likely because speed is the most directly affected feature when stopping, and a plane typically remains at low speed for several frames while stopped. Once speed was removed, performance dropped, but the shift forced the models to rely on more meaningful relational features.

New feature engineering steps were introduced:

- **Priority Encoding:** For the observing plane, the feature is 1 if it has priority (i.e. it's closest to its goal); otherwise, one of the other two planes gets a 1 based on who's nearest, and the rest are 0.
- **Projected Distance:** A directional filter that encodes both relative angle and distance into a single scalar value: $\text{projected distance} = d \cdot \cos(\theta)$. This feature becomes smallest when another agent is directly in front and close, which is exactly when yielding is required.

Eventually, the best performance was achieved using only the eight features of the two nearest agents, hence removing the observing agent's own state entirely. These features were:

- Speed
- Projected distance
- Relative angle
- One-hot priority

5.5.2 Results and Integration

Model	Accuracy	Notes
Logistic Regression	69.42%	Weakest performer; confusion matrix showed high false positives (see Figure 12.B in Appendix)
Decision Tree	76.30%	More balanced; most importance placed on relative angle and projected distance.
Random Forest	89.35%	Best overall accuracy; distributed feature importance and lower variance.

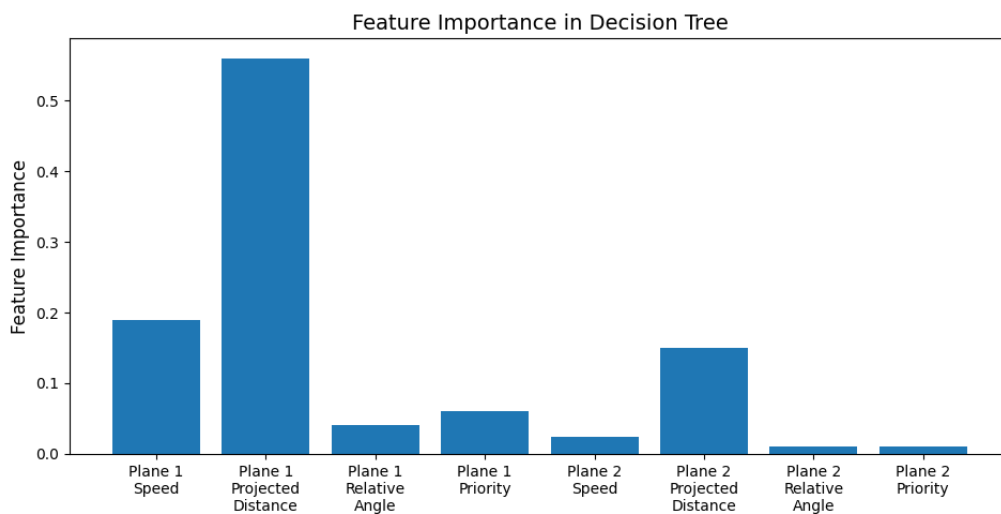


Figure 5.G: Feature importance matrix from the final Decision Tree model.

When deployed in the simulation, the ML classifiers behaved more conservatively than any MARL-based models. This made for a simulation with agents that never progressed whatsoever and remained stationary after the pushback phase. Attempts to integrate the engineered features into MARL showed observable improvements, suggesting the issue in MARL wasn't necessarily the input encoding but the reward structure and training dynamics. Ultimately, ML classification proved useful as a diagnostic tool, but it was not viable as a final control solution due to its lack of temporal awareness and inability to adapt to dynamic scenarios.

5.6 Final Implementation and Trade-offs

After several weeks of intensive experimentation, none of the machine learning or multi-agent reinforcement learning approaches yielded any reliable or usable models. Although an LSTM-based method had already been implemented earlier and showed promising results, it was initially dismissed due to its limited capabilities as it was only trained on Birmingham Airport's taxiway structure and thus could not generalise to other layouts.

Another limitation emerged after the go-around functionality was introduced into the simulation. The LSTM had never seen the go-around behaviour during training, thus producing inaccurate predictions as it tried to constrain the predictions to the bounds of the taxiways (see Figure 12.C in the Appendix). However, the LSTM approach proved to be effective for collision avoidance. The system works by continuously predicting the positions of all aircraft at future timesteps. If any predicted positions overlap, a priority check is done based on distance to the final waypoint on the ground. The aircraft with lower priority (farther distance) is instructed to yield (by braking), while the other continues. This coordination logic successfully eliminates all collisions in the simulation and allows for effective coordination.

After empirical experimentation, the threshold distance for triggering a yielding instruction is 36 pixels. Considering each aircraft is approximately 17 pixels wide, this ensures at least 20 pixels (about 50 metres) of clearance is given before executing the instructing. This provides enough time for a plane to decelerate and stop, and in practice, this maintained a separation of at least 17 pixels, about 45 metres, between planes when queueing. This corresponds closely with real-world safety standards, where 50 metres of separation is considered the minimum safe distance [64].

While the final solution does not generalise beyond the environment it was trained on, it demonstrated that LSTMs are still highly capable for coordination with enough context and training. Within Birmingham Airport, it consistently produced accurate, reliable, and safe behaviours, proving to be the most effective and practical solution for this simulation. It acts both as a system to resolve intersection conflicts and allows the simulation to re-create queueing behaviour.

6 Runway Monitoring and Management

6.1 Overview

At this stage, there is a simulation with intelligent agents that can move around the airport autonomously, as well as an LSTM-based model that helps resolve conflicts at intersections and enables aircraft to stop when queueing. Queueing can be described as an inherent behaviour like how a human pilot would naturally maintain distance from a plane in front. The conflict resolution aspect more closely resembles the air traffic controller's role where one aircraft is instructed to yield to another.

These behaviours can be differentiated between by looking at their aircraft's paths. If two aircraft are approaching the same point from different branches of the taxiway network, then the situation requires an intelligent conflict resolution. On the other hand, if the aircraft are following the same route, or one route is a subset of the other, then it would be a case of queueing and not decision-making. In this simulation, the LSTM-based model provides us with both behaviours. As discussed in the previous chapter, the limitation of this approach is that the LSTM is only trained on Birmingham Airport, meaning it won't generalise to other airport layouts.

In the simulation thus far, one major area of unresolved conflict is runway access, namely when a landing aircraft is on final approach while another aircraft is waiting to take off. The original goal was to solve this using Hierarchical Reinforcement Learning (HRL). If reinforcement learning had succeeded earlier, then HRL would have been a natural extension since different models could be assigned to different layers of the decision-making process [65], but since reinforcement learning failed to learn coordination in the simpler task, HRL was abandoned.

Instead, a more straightforward alternative was explored inspired by real-world ATC. In air traffic control, humans determine runway occupancy either by looking out of the tower or using a radar screen. To replicate this behaviour, it was decided to implement a computer vision model to scan the radar display and identify whether a plane is currently occupying the runway or not and thus be able to coordinate between departing and arriving aircraft and ensure no runway incursions occur.

6.2 Approach and Dataset Generation

Because computer vision is a supervised learning method, a labelled dataset needed to be created. A Python script was written to automatically generate an image dataset from the simulation. Using Pygame, 1200 by 90 pixel images were created capturing a snapshot of the radar screen with the Birmingham Airport background and a randomly placed aircraft image. Aircraft were spawned in and around a predefined region which was the runway and surrounding taxiways (only up to the holding points), as this is the region that should only be occupied by one aircraft at any time and needed to be monitored. This included positions where only parts of the aircraft (e.g., wingtip or tail) entered the detection zone, creating some edge cases.

To ensure visual diversity and reduce the likelihood of overfitting to Birmingham’s runway layout or markings, each image was manually augmented with changes in brightness, contrast and adding rotations and noise. 2,500 images with planes were generated, and another 2,500 were generated with no plane within the radar area, totalling 5,000 images (~ 1.1 GB). The images were saved as PNGs, with corresponding bounding box labels stored in a CSV file. The bounding boxes were normalised to the size of the radar frame (scaled between 0 and 1). The dataset was then split into training and validation sets using an 80/20 ratio.

6.3 Model Training and Performance

For this task, it was decided that simple classification would not be fully sufficient, as in many cases it would also be important to know where a plane is and not just whether it is present (although in this implementation localisation was not necessary). For this reason, the computer vision model needed to include localisation capabilities, and the dataset was labelled accordingly from the beginning.

The goal was to use an extremely lightweight, accurate, and fast model that could reliably detect aircraft without introducing latency. While several localisation models were considered (e.g. Faster R-CNN), the YOLO (You Only Look Once) family was chosen for being the fastest option and maintaining great accuracy [66]. YOLOv8 was selected and trained using the Ultralytics Python package. This was a single-class object detection task, since each image either contained a plane or did not (and this was the only target class).

The model was trained on the synthetic dataset for 50 epochs over the course of an hour. Performance metrics were exceptionally high:

- Precision: 1.00
- Recall: 1.00
- mAP@0.5: 0.995
- mAP@0.5:0.95: 0.995

These results suggest that every predicted bounding box matched a ground truth box, and every actual aircraft was detected. While the scores are excellent, they do reflect the simplicity of the task since the planes are high-contrast objects on the background and the validation set was generated under the same augmentation conditions as the training set. A noteworthy result is that even partially visible aircraft like those with only the wing visible from the edge of the radar area were correctly detected. While it is safe to say that this dataset lacks the variability to generalise to real-world conditions, the task is both straightforward and well-suited to CV within the scope of this simulation.



Figure 6.A: Example outputs on validation frames. These results were generated automatically during training.

When integrated into the simulation, the model completed inference decisions in under 100ms, with highly reliable visual outputs (although not as perfect as seen during validation). The model was able to consistently track aircraft in the target area, but it was configured to run once every 0.5 seconds to balance performance and efficiency, as it was still computationally expensive to run every frame. In some frames, multiple and overlapping bounding boxes appear, but this does not affect the accuracy of the simulation. A small issue occurred early on where the model sometimes misidentified runway markings as aircraft, likely due to a lack of variation in the training set's visual patterns. To fix this, the simulated input images were modified slightly with the same augmentation techniques used to create a dataset to suppress the prominence of runway markings. This removed false positives and stabilised predictions more.

6.4 Integration into Runway Coordination Logic

With a system now in place that can reliably detect whether an aircraft has entered a controlled area, we can effectively control which aircraft are allowed to enter or cross them. Previously, the nodes in the taxiway network included a `holding_point` flag, but since there was no logic implemented to manage access, it wasn't enforced. Now, that logic has been fully implemented. In the updated system, any aircraft approaching a holding point must be granted explicit authority before it can proceed. If the aircraft's `authority` flag is not set to `True`, it will stop and hold at the point. Once permission is granted, the flag is set to `True`, allowing the aircraft to pass. After passing through the holding point, the `authority` flag is reset.

To determine which plane gets permission to enter the runway, several common scenarios must be handled:

- The runway is empty.
- A plane is waiting to take off.
- A plane is on final approach for landing.
- Both a landing and departing aircraft want to access the runway.

In real-world ATC, the controller would make these decisions based on many factors like fuel levels of aircraft, delays, workload, and weather. In this implementation, a simplified priority-based system was developed. Each aircraft is assigned a priority variable, which increases over time, and this was set to the following based on empirical observations:

- +0.01 per frame while on the ground.
- +0.04 per frame while airborne.

This variable acts as a representation for urgency or waiting time. Over time, an aircraft's priority reflects how long it has been waiting or how "desperate" it is to proceed. Although it is simple, this mechanism can be scaled in ways such as increasing an aircraft's priority rapidly in the event of an emergency.

As an example of this mechanism, if a runway is clear and both a departing and landing aircraft are queued:

- The landing aircraft, having accumulated more priority while airborne, will typically be given access first.
- The departing aircraft will be told to wait.
- If, however, the ground aircraft has been waiting for a very long time, it may eventually surpass the landing aircraft in priority. At that point, the landing aircraft will be instructed to go-around.

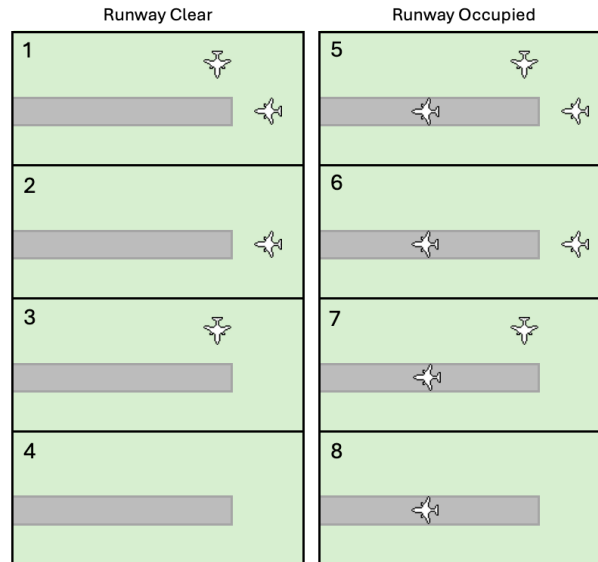


Figure 6.B: Eight possible runway access scenarios handled by the priority-based logic. Left: Cases where the runway is clear. The system compares priorities to decide whether a landing or departing aircraft proceeds. Right: Cases where the runway is occupied. Any additional aircraft are either instructed to hold (ground) or go-around (airborne), ensuring runway safety is maintained.

This logic is actively supported by the CV model, which checks if the runway is indeed clear. If a plane prematurely enters the runway, the model will detect it and issue an immediate go-around command to the immediate landing aircraft. While the implementation of the CV system is simple, it allows for extensions to be made to monitor other obstacles, such as airport vehicles. Although the current model is only trained to detect aircraft, it could be retrained or expanded to recognise other classes of objects. If an unauthorised vehicle or object enters the controlled zone, the CV model would detect it and trigger a go-around.



Figure 6.C: Snapshot of the simulation showing aircraft detection in practice.

This system effectively simulates the decision-making of a human air traffic controller with all its flexibility, but without requiring hardcoded logic for every scenario. In fact, this CV-based safety layer may even catch rare or subtle situations that a human might overlook, making it a reliable real-time monitor of runway safety.

7 Pilot Request Intent Recognition

7.1 Overview and Approach

At this stage, the simulation environment functions well, but not exhaustively, and there is plenty of room for expansion. However, since it is fully simulated, there is currently no mechanism for real-time interaction. In a real-world deployment, there would need to be some form of interface between the pilots in their aircraft and the simulation with its logic. The aim was to replicate the real-world ATC workflow, where the controllers interpret a pilot's request and act accordingly, like work done by Kleinert et al. [30]. These requests would then trigger changes in the simulation, such as an aircraft leaving the stand, taxiing to the runway, or requesting permission to land.

This falls under the field of natural language processing (NLP). Here, the task is to receive a text request from a pilot and classify it based on intent. There are popular language models such as GPT (Generative Pretrained Transformer) [67] and BERT (Bidirectional Encoder Representations from Transformers) [68] which are commonly used for such tasks.

For this task, it was decided to use the BERT language model. It is a transformer-based language model developed by Google that processes input bidirectionally, meaning that it looks to the left and right of each word to extract context. BERT is pre-trained on large text corpora using masked language modelling and next sentence prediction, making it specialised for understanding intent in structured texts. Compared to GPT which is unidirectional and focuses more of generative tasks, BERT is better suited for classification tasks, especially in short and structured communication like aviation phraseology.

English is the international language of aviation, and as such, there is a globally standardised phraseology, such as specific words that must be said depending on the request and always mentioning the aircraft identifier. However, this standard is not always adhered as it is common for pilots to revert to their native language or use broken English, which introduces noise and irregularity into the communication [29]. These inconsistencies can lead to misunderstandings and errors in both human and machine interpretation.

For this NLP task, the use of a dataset is to be expected. Although open-source datasets like the ATCO2 corpus, mentioned in Section 2.4.2, offer real-world pilot-controller communications, they were not used in this project. The dataset is large, noisy, and contains a wide range of speech acts beyond the limited set of contexts relevant here. Manually extracting the necessary examples would be highly time-consuming, particularly as the annotated labels are not freely available and are locked behind a paywall.

7.2 Initial Implementation and Limitations

The first attempt at this task involved generating a dataset of 200 text-based requests, labelled across four categories with 50 of each type: pushback (class 0), taxi (class 1), take-off (class 2) and landing (class 3). These 200 examples were generated using a large language model. They

were clean, standardised, and conformed closely to official aviation phraseology. The BERT model was fine-tuned on this dataset and achieved 100% validation accuracy.

While this initially seemed promising, further testing showed that the model did not generalise well. When fed with actual speech-to-text data which inevitably contained small errors or informal phrasing, its performance was very poor. The issue was that the training dataset was too clean and predictable. Key action words were often explicitly present, and the model simply learned to pick those up, like simply doing a substring search.

Therefore, a more robust, alternative approach was needed. The goal was expanded for the next implementation to include both classifying intent from noisy or ambiguous request phrasing and extracting the aircraft identity (i.e., the source of the request) based on flight identifiers. This setup is more reflective of real-world ATC interactions and workflows, where controllers must not only understand the command but also identify who is speaking, particularly given that in typical traffic, around 20 aircraft will be communicating on a single channel in a small international airport.

7.3 Dataset Preprocessing and Augmentation

7.3.1 Dataset Expansion and Cleaning

The obvious first step to improve the model was to expand the dataset. As before, a large language model was used to generate the new data, but this time the prompt was changed to ask for longer and more unconventional valid pilot requests. These covered variations in phrasing that might be found in real speech from, for example, non-native speakers. Since this version also aimed to support entity extraction, a new column was added to include the aircraft identifier for each request.

Before any performing any augmentation, the text needed to be processed to resemble real-world speech-to-text output, which typically has no punctuation, has all letters in lowercase, and write numbers as words instead of numerals. To replicate this behaviour with the dataset, all punctuation was removed, any numerical values (e.g., flight numbers like “101”) were converted to text (e.g., “one zero one”), and all text was converted to lowercase. This was applied through all the fields in the dataset, namely the request text and the aircraft identifier for consistency.

7.3.2 Data Augmentation: Homophones, Rhymes, and Splits

To introduce ambiguity and mimic the kind of errors and distortions that occur in real-life speech-to-text transcriptions, a multi-staged augmentation process was implemented where several types of Augmentation were done:

- **Homophones:** Words that sound the same but have different meanings or spellings (e.g., “to”, “too”, “two”).
- **Rhymes:** Words with similar ending sounds (e.g., “hat”, “cat”, “bat”).
- **Term splitting:** Aviation-specific terms like “takeoff” might be split into “take off” to simulate incorrect parsing.

To perform these augmentations, homophones were scraped using a Python script which made a request to homophone.com [69], an online database of all know homophones for words in the English dictionary. Rhymes were found via a script which made requests to the Datamuse API [70]. These augmentations were inspired by prior work in text augmentation for NLP tasks [71].

Each row in the dataset was duplicated, and the augmentations were applied to each word in a request with probabilities:

- 50% chance of applying a homophone substitution.
- 20% chance of applying a rhyme (if homophone wasn't used).
- 60% chance of applying a split (if neither homophone nor rhyme was applied).

This process added some ambiguity to the requests, which increases its realism and difficulty.

7.3.3 Candidate Generation for Entity Recognition

To simulate flight identification, a column of potential candidates was introduced in the dataset. This was done as the model would need to be aware of existing aircraft on a channel, then pick the one it thinks most likely made the request. This column included:

- The correct flight identifier for the request.
- At least one similar flight with the same airline name and a 20% chance of including a second one.
- Additional random flights from other airlines to have a total of five candidates.

As with the request text, the list of identifiers was processed to replace numerals with words.

7.3.4 Final Dataset Structure

Since the original 400 samples were duplicated and the duplicates were augmented, this resulted in a dataset of 800 rows, with 400 examples being clean and the other 400 being noisy. Each row in the dataset included the request text, the aircraft identifier, 5 candidate flights (with one being correct) and the intent label of the type of request.

Stage	Request
Original	Gatwick Ground, EasyJet 101, request taxi.
After processing	gatwick ground easyjet one zero one request taxi
After augmentation	gatwick sound easyjet gun xero one request braxy

Table 7.A: Example transformation of a taxi request from the dataset.

Request	gatwick sound easyjet gun -xero one request braxy
Flight	easyjet one zero one
Label	1 (taxi)
Candidates	['alaska five two two', 'gol six zero six', 'easyjet one three eight', 'easyjet one zero one', 'aelta four seven six']

Table 7.B: Example datapoint from the dataset.

To train the models, a random 90/10 split was used for training and testing. This allowed for there to be enough variation in training and evaluate on unseen, equally difficult, samples.

7.4 Request Classification Using BERT

Similarly to the initial approach, BERT, specifically the `bert-base-uncased` model, was fine-tuned using the HuggingFace Trainer API [42]. It was trained on the augmented training dataset containing the four intent classes. Although training was initially set for 10 epochs, the validation loss plateaued after 6 epochs, at which point the training was run again and was stopped at 6 epochs. The full training process took approximately 15 minutes.

When evaluated on the test set, the model achieved an accuracy score of 100%, correctly classifying every request. All the key evaluation metrics; precision, recall, and F1-score were 1.0 for every class, and the confusion matrix showed a perfect diagonal, as seen in Figure 7.A.

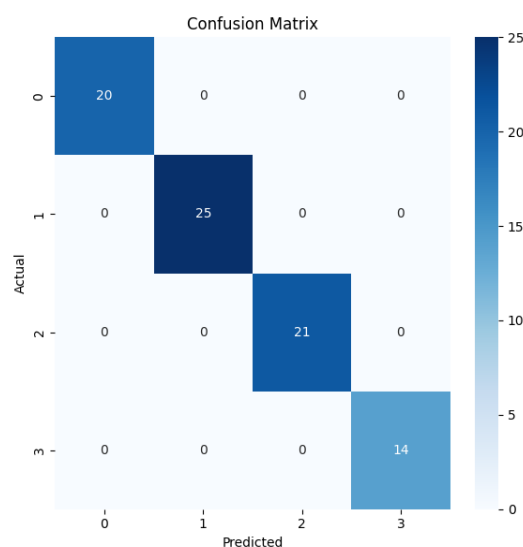


Figure 7.A: Confusion Matrix of BERT intent classifier on the augmented test set.

By contrast, the earlier version of the model (trained on the clean, unaugmented data) achieved a test accuracy of only 86.25% on the same test set. This drop in performance is likely attributed to the older model’s reliance on explicit keywords like “landing” and “takeoff”, which failed when tested against augmented variants such as “braxy” used in place of “taxi”. Overall, the final model generalised very well, even when trained on a heavily augmented and noisy dataset.

7.5 Flight Entity Identification and Candidate Ranking

7.5.1 Initial Attempt Using Fuzzy Matching

As a lightweight first approach, fuzzy string matching was explored to assess how reliable a simple, lightweight approach would be. Fuzzy string matching uses the Levenshtein distance metric [72], which measures how different two strings are. It calculates the minimum number of single-character edits (insertions, deletions and substitutions) required to change one string into another. The goal was to compare the request string to each candidate flight identifier and select the one with the highest similarity score.

In practice, this worked reasonably well on instances where identifiers were clean and intact, but performance dropped significantly when run on augmented inputs. Each request was compared to all candidates, and the most similar string (based on character-level distance) was selected. This was run on the augmented test set generated previously and the evaluation results include top-k accuracy and mean reciprocal rank (MRR) scores:

- Top-1 Accuracy: 0.3122
- Top-3 Accuracy: 0.7385
- Mean Reciprocal Rank (MRR): 0.5598

Top-k accuracy measures how often the correct label is found within the top k predicted results, rather than requiring it to be the single highest prediction (e.g., top-3 measures how often the correct label is present in the top three rankings generated by the model).

The MRR score is calculated as:

$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i}$$

Where:

- $|Q|$ is the total number of queries
- rank_i is the position of the first correct result for the i^{th} query

These metrics could be interpreted as percentages, and ideally, we would like them to be as close to 1.0 (100%) as possible. However, these results clearly showed that fuzzy matching struggled with the noisy data. While it is conceptually simple, it was discarded as it was deemed unreliable for realistic speech-to-text use cases.

7.5.2 Ranking-Based Identification with DistilBERT

In an attempt to improve accuracy, the task was reframed as a ranking problem using a language model. Since this required evaluating multiple input pairs quickly, a lightweight model, DistilBERT, was chosen. DistilBERT is a faster, smaller version of BERT that retains about 95% of its accuracy while being 60% smaller and significantly faster [73].

Transformers like BERT are not capable of ranking by default, so the problem was reformulated as a binary classification task [74]:

- Each training sample was a pair: (request_text [SEP] candidate_flight)
- The model predicted a score (logit) representing how likely the candidate was the correct one
- For each request, candidates were ranked by their logits
- Candidate with the highest logit is picked as the models chosen entity

Training followed the same structure as the intent classifier previously trained. DistilBERT was trained on the augmented training dataset and ran for 5 epochs, which took just under 15 minutes. Overall, the training was stable, and loss consistently decreased until plateauing at epoch number 5 (as seen in Figure 7.B).

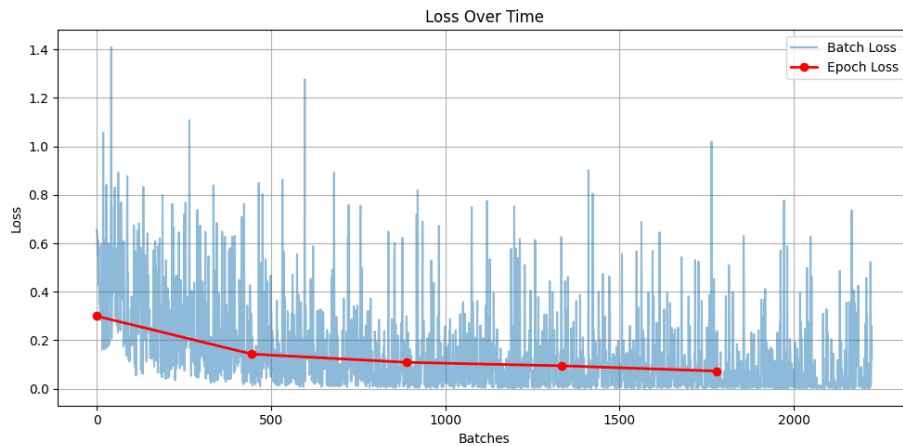


Figure 7.B: Training loss curve for the DistilBERT ranking task.

When evaluated using the same previously used metrics on the test set, the results were:

- Top-1 Accuracy: 0.9625
- Top-3 Accuracy: 1.0000
- Mean Reciprocal Rank (MRR): 0.9792

These results were a massive improvement over fuzzy matching. Initially, the plan was to use Levenshtein as a filtering step (to reduce the candidate size before passing it to BERT), but this proved to be completely unnecessary. DistilBERT handled the full ranking task on its own, delivering fast and highly accurate predictions even on noisy, augmented input.

7.6 Practical Experimentation

To evaluate the model in a practical setting, it was tested using real speech-to-text input, as there is not practical setting of integrating it into the simulation. A Python script was written to randomly generate requests not included in the training or testing datasets. These were intended for testing on real users, with the requests displayed in a Terminal window and read aloud by participants. Apple's built-in speech-to-text dictation on a MacBook was used to transcribe the spoken input. The transcription process had a word error rate (WER) of 38.92%, reflecting the challenge of accurately capturing spoken aviation phraseology. The transcribed text was hidden from the users to prevent them from correcting themselves or speaking unnaturally slowly to aid the transcription process. To ensure fairness, the test included participants with different English language backgrounds and accents. Across five participants and 25 test cases, the model achieved 92% accuracy, with only one misclassification and one incorrect entity extraction. In both cases, however, the correct prediction was ranked second (results detailed in Section 0 of the Appendix).

8 Project Management

This chapter outlines how the project was researched, planned, and developed. It presents the timeline of work, including an initial project concept and the decision to pivot away from it, and highlights the methodologies, tools, and techniques used to manage the project across its lifecycle.

8.1 Project Timeline

8.1.1 Initial Project Timeline

Although not the final project, the initial concept for this report formed part of the academic journey and influenced time planning and learning. The prior planned project aimed to reconstruct visual data from EEG brain signals using generative models. However, after preliminary research and experimentation, the scope proved unfeasible within the limited timeframe and available resources. A change in direction was made in late December.

Phase	Activity	Start Date	End Date
Conceptualisation	Project Specification Draft	10/10/2024	15/10/2024
Literature Review	Brain Signals & EEG Analysis	16/10/2024	31/10/2024
Technical Research	Image Generation Techniques	01/11/2024	20/11/2024
Expert Consultation	Discussions with PhDs and Academics	21/11/2024	27/11/2024
Dataset Exploration	Evaluation of Open EEG Datasets	01/12/2024	10/12/2024
Feasibility Review	Project Scope Evaluation	10/12/2024	14/12/2024
Project Pivot	Search for New Direction	15/12/2024	24/12/2024

Table 8.A: Initial Project Task Schedule.

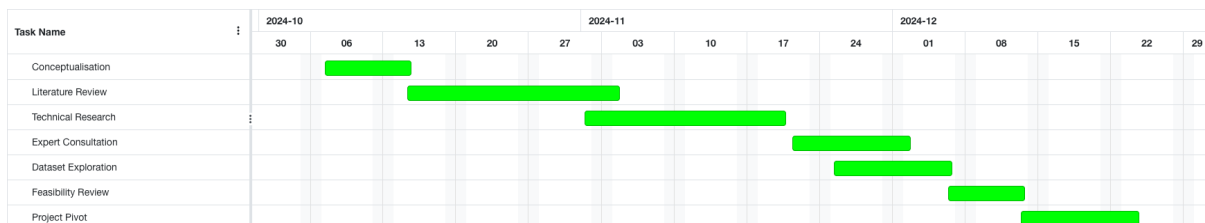


Figure 8.A: Initial Project Timeline Gantt Chart.

8.1.2 Final Project Timeline

This new project focused on developing a proof of concept for an autonomous air traffic control system using deep learning. The scope included research, simulation design, and iterative model development. While the actual timeline deviated slightly from the original plan due to a growing focus on specific components of ATC automation, the overall structure remained largely intact. A minor delay of approximately one week occurred as additional time was allocated to overcoming challenges in implementing certain AI models. All activities were scheduled with the project demonstration (12/03/2025) and final submission deadline (17/04/2025) as key milestones.

Phase	Activity	Start Date	End Date
Ideation & Planning	Final project idea selection and planning	24/12/2024	03/01/2025

Literature Review	Research on ATC systems, CV, RL, MARL, LSTM, NLP, GNN	04/01/2025	21/01/2025
Environment Setup	Simulation design and dataset preparation	15/01/2025	28/01/2025
RL Model: Plane Movement	Reinforcement learning for basic aircraft taxiing	29/01/2025	06/02/2025
MARL & LSTM: Conflict Resolution	Multi-agent coordination and temporal planning	07/02/2025	28/02/2025
CV: Runway Management	Computer vision model for runway occupancy & control	01/03/2025	07/03/2025
NLP: Pilot Speech Modelling	Natural language processing for communication simulation	08/03/2025	12/03/2025
NLP Revision	Complete overhaul of NLP system post-feedback	13/03/2025	18/03/2025
Refinement & Evaluation	Cleanup, fine-tuning, additional experimentation	19/03/2025	26/03/2025
Thesis Write-Up	Complete thesis writing and documentation	27/03/2025	17/04/2025

Table 8.B: Final Project Task Schedule.

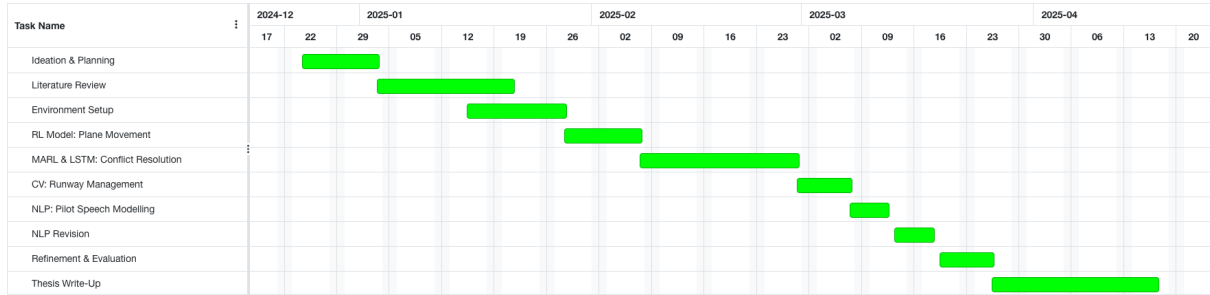


Figure 8.B: Final Project Gantt Chart.

8.2 Development Methodology

The development of this project followed a largely linear methodology since most components are dependent on previous components. This is due to the nature of the AI components and their dependencies within each other in the simulation. Unlike traditional iterative approaches like Agile [75] that rely on cycles of development and user feedback, this project required each module to be developed in a sequential manner, where the successful completion of one stage was a prerequisite for the next. There are also no methods of testing or feedback as the only things aspects that can be evaluated at each stage are how well models perform individually.

The process began with research to inform the implementation of each subsystem. Once a large enough understanding was gained on a topic, a simulation environment was constructed to support the development and testing of AI agents. As each component was validated, the simulation environment was incrementally expanded to accommodate the requirements of the next subsystem.

AI model development itself was guided by an iterative approach. Initial implementations were evaluated through performance and behaviour within the environment. If a model failed to meet the expectations, then a considerable amount of effort was put into debugging and fine-tuning.

In cases where issues continued to arise, alternative approaches were explored through different research, leading to revised or entirely new implementations.

8.3 Tools

Local Storage and Git – All code was developed and stored locally, with Git used for version control. Regular commits were made to a remote repository to track changes across the project’s lifecycle. This enabled the ability to revert to earlier versions, compare model performance across checkpoints, and maintain a reliable development history—particularly useful during iterative AI experimentation.

Microsoft 365 Applications – Microsoft Word, PowerPoint, and Excel were used extensively for documentation and organisation. Excel was used to track model performance metrics and hyperparameter values during tuning phases. Word and PowerPoint supported the preparation of the final thesis document and project demonstration materials.

Google Workspace (Colab and Drive) – Google Colab was used to leverage cloud-based GPU acceleration for training deep learning models, which significantly reduced processing time compared to local hardware. Google Drive was used for storing large datasets and for transferring files across machines during different development stages.

Visual Studio Code (VS Code) – All code for the simulation environment, AI models, and supporting scripts was developed using VS Code. The IDE’s rich extension ecosystem allowed integration with Python-based tools for code linting, debugging, and visualisation. Its flexibility and performance made it well-suited for managing complex multi-component systems.

OnlineGantt – OnlineGantt is an online tool that was used to create interactive Gantt charts for project planning and progress tracking. Its drag-and-drop interface facilitated easy updates to task timelines as the project evolved.

Adobe Photoshop – Photoshop was used for creating project visuals, including diagrams and simulation visuals. It was also employed in early prototyping stages to estimate pixel dimensions and spatial layouts within the simulation environment.

Nvidia RTX 4090 – A powerful GPU was used for a subset of the training tasks required for this project. It was used specifically when training the PER implementation of MARL (Section 5.4.3) and the YOLO object detection model (Section 6.3). Access to this hardware was limited and time-restricted, so its use was focused on parts of development where performance gains were most critical.

9 Evaluation

The objective of this section is to assess how well the system fulfils its intended functional goals, performance criteria, and practical viability as a proof of concept for autonomous air traffic control. It also discusses limitations, shortcomings, and evaluates the individual components that contributed to the overall system.

9.1 Summary of Requirements

One of the main measures of success for this project was the fulfilling of the functional requirements defined for the system environment and the intelligent air traffic control subsystem, as outlined in Chapter 3. The results of this are summarised in Table 9.A, where a (✓) indicated the requirement is met, a (–) indicates it was partially met, and a (✗) indicating it was not met.

Functional Requirement	Outcome
FR1	✓
FR2	✓
FR3	✓
FR4	✓
FR5	✓
FR6	✓
FR7	✓
FR8	✓
FR9	✓
FR10	✓
FR11	✓
FR12	✓
ATC-FR1	✓
ATC-FR2	–
ATC-FR3	✓
ATC-FR4	✓
ATC-FR5	✓
ATC-FR6	–
ATC-FR7	✓

Table 9.A: Summary of Functional Requirements Fulfilment

As shown in Table 9.A, all simulation environment functional requirements (FR1–FR12) were fully met. However, ATC-FR2 and ATC-FR6 of the intelligent ATC requirements were only partially met.

ATC-FR2: “The system shall prevent multiple aircraft from occupying the same runway or conflicting taxiway segments simultaneously.”

This requirement was partially met. The system reliably prevents multiple aircraft from occupying the same runway at the same time. It also resolves taxiway conflicts, particularly when aircraft are navigating intersecting or diverging taxi paths. However, it fails to resolve

head-on conflicts where two aircraft enter the same taxiway from opposite directions. In such cases, a deadlock occurs, which is due to the absence of a conflict resolution mechanism for coordinating taxiway entry like what was done with the runway.

ATC-FR6: “The controller shall coordinate actions across all aircraft centrally, though the architecture may also support decentralised alternatives.”

This requirement was also only partially satisfied. While most decisions are coordinated via the central simulation loop, the architectures experimented with and developed throughout this project did not align fully with the originally intended design. Although all developed intelligent components proved to be reliable and robust, the lack of a fully explored central coordination mechanism means that this requirement cannot be considered fully met.

9.2 System-Level Behaviour and Findings

Evaluating the performance of an autonomous ATC system is a challenging task, since the goal is to actively avoid unsafe situations before they occur. As a result, there is no quantifiable metric such as “collisions avoided” to assess the system’s performance. Instead, evaluation would have to be based on a combination of component metrics and outcomes of simulations under varying conditions [35]. During the development of the RL components, individual models were assessed using standard reward-based metrics. These scores were fine-tuned through empirical observation over many training runs, with quality of the model evaluated through a visual inspection of the resulting behaviours. Notably, the supervised ML tasks performed well quantitatively but, in practice, were worse than their MARL counterparts, as discussed in Chapter 5.

The simulation framework itself was iteratively improved based on observed conflicts. For example, the initial versions of the simulation environment allowed landing and departing aircraft to access the runway simultaneously, a behaviour that required the implementation of CV within the system, as discussed in Chapter 6. One practical measure of the system’s effectiveness is whether it can maintain safe and efficient operations under realistic air traffic volumes. Based on historical flight data, Birmingham Airport handles between four and thirteen arrivals and departures per hour. It is important to note that this figure reflects scheduled flights, and it is likely that human controllers are qualified to handle more traffic than what the airport gets. Nevertheless, it was used as a benchmark for simulation testing.

To simulate realistic traffic flow, the Bernoulli distribution of a pushback or landing occurring were set to a low success probability (1 in 10,000 frames), whereas during development, this was set to a higher success probability of around 1 in 2000 frames. A test scenario was constructed with 15 departures and 15 arrivals active within the simulation. The system successfully completed all required operations over a 32-minute period without any incidents, demonstrating that the autonomous controller could safely handle traffic levels comparable to those seen at Birmingham Airport.

A more demanding infinite simulation test was also conducted. In this test, aircraft were set to continuously cycle in a manner like that shown in Figure 3.E. What this entails is that after

landing and reaching a stand, an aircraft would wait for a short period before initiating a new departure. The system was stressed further by increasing the frequency of landing and departure events. It was during this high-load scenario that a limitation was found. Aircraft travelling in opposite directions on the same taxiway segment could encounter each other head-on, resulting in a deadlock. This blocked further movement and halted the simulation, as no rerouting or yielding mechanism had been implemented for this scenario. This behaviour relates directly to the previously discussed partial fulfilment of ATC-FR2 in Section 9.1.

Early on during the implementation stage of this system, various solutions were considered for this limitation, including the use of graph neural networks (GNNs) to model taxiway conflicts [76]. While the graph-based structure of the airport would make this a logical approach, it was abandoned due to concerns about generalisability over different environments and limited project time. An alternative approach was considered using trajectory prediction via an LSTM with longer temporal predictions, however this was not implemented.

Instead, an algorithmic approach was considered to predict and avoid directional conflicts on shared taxiways. A general form of this algorithm is provided below.

```

For each aircraft A in simulation:
  Get route  $R_A$  across taxiway segments
  For each time step  $t$  in future time window:
    Estimate position  $P_A(t)$  based on average taxi speed
    Determine occupied taxiway segment  $T_A(t)$  and direction  $D_A(t)$ 

For each taxiway segment T:
  If two or more aircraft plan to occupy T at the same time:
    If directions conflict (e.g.,  $D_A(t) = +1$  and  $D_B(t) = -1$ ):
      Identify conflict

      Resolve conflict using one of the following:
        - Yielding rule (e.g., based on priority)
        - Rerouting one aircraft using modified Dijkstra's algorithm

```

Algorithm 3: Predictive Taxiway Conflict Resolution

While this algorithm is simple and assumes there is knowledge of an aircraft route and speeds, the approach does not account for non-deterministic factors such as traffic congestion. To function effectively in a real-world scenario, it would require heuristics or learned models to estimate delays and whether rerouting is possible. Additionally, while rerouting logic is not currently implemented, it could be achieved using a standard graph traversal algorithm such as Dijkstra's, with the only modifications needed are to exclude congested segments of taxiway.

Within the system, there is a large room for improvement. The simulation is by no means exhaustive and lacks several key mechanisms, such as global conflict resolution over shared taxiway sections, a rerouting strategy once a deadlock occurs, and overflow logic for when airport stands are full (in the simulation, aircraft simply stop landing, which is unrealistic). While every effort was made to design the models to be as generic as possible, this was not always achievable. Certain aspects of the system appear to be overfitted to Birmingham Airport, and no cross-airport generalisability was tested.

9.3 Component Evaluation

Throughout this project, extensive experimentation was done with various AI models and architectures, along with the ideation of their roles within the system. While model-specific evaluations and metrics were discussed in earlier chapters, this section gives a final summary and assessment of the models that were integrated into the final simulation.

Reinforcement Learning for Aircraft Movement – Although not part of the intelligent ATC decision-making component, this model plays a critical role in simulating the non-deterministic dynamics of real-world aircraft movement. The RL agent was able to accurately maintain realistic taxiing speeds, turning behaviour, and adherence to taxiway centre lines. The deviation from the centre line was minimal, remaining within a tolerance of approximately four pixels. This greatly enhanced the realism of the simulation and contributed to the believability of aircraft behaviour under autonomous control.

LSTM for Conflict Management – The LSTM model proved to be effective in predicting aircraft positions and resolving conflicts at intersections. It was particularly reliable when combined with rule-based logic for conflict resolution. However, the model was trained exclusively on data collected from Birmingham Airport, resulting in a degree of overfitting. As a result, it struggled to predict less common manoeuvres, such as go-arounds, which were not present in the training dataset. Nevertheless, the model is relatively straightforward to reproduce and could be retrained for other airport layouts or expanded datasets with minimal additional effort, making it a suitable tool for broader applications.

Computer Vision for Runway Management – The YOLO model performed well on its evaluation set, achieving high scores. The task itself was relatively simple, but the model demonstrated strong potential to be expanded further to detect a wider range of airport obstacles. In hindsight, the dataset could have benefited from greater diversity, including less reliance on augmentation and more varied backgrounds and aircraft images. These limitations may have contributed to a slight drop in performance when deployed in the live simulation. Nonetheless, the model’s performance remained acceptable, and any minor discrepancies were manageable through code-based filtering and adjustment.

Natural Language Processing for Pilot Request Analysis – The initial NLP implementation was found to be unrealistic and inefficient, but significant improvements were made in the final version. The refined model demonstrated high accuracy in understanding pilot requests and extracting identifying information. Although this component was not integrated into the real-time simulation, it was evaluated separately using human voice inputs, where it performed consistently with earlier test results. This approach has considerable potential for future integration, especially as the scope of the simulation expands to include more complex scenarios beyond taxiing, take-off, pushback, and landing.

10 Conclusion and Discussion

This chapter summarises the achievements of the project and the work completed and reflects on the impact of the project and the future work in this field that is to come.

10.1 Summary of Achievements

In this project, a lightweight simulation of Birmingham Airport was developed, incorporating realistic taxiway layouts, accurate scaling, and simplified aircraft dynamics to reflect the real-world. Although not all operational procedures like scheduling and sequencing were replicated, efforts were constantly made to maintain a high degree of realism throughout. Multiple DL models were integrated into the simulation to explore the potential for automating aspects of ground-based ATC, with the goals of improving safety, throughput, and efficiency. While many models were explored, none were intended to be exhaustive or fully generalisable.

The final system successfully demonstrated aircraft movement using RL, establishing that deep RL agents can navigate taxiways while maintaining adherence to realistic constraints. Furthermore, an LSTM model was employed for trajectory prediction, allowing for a form of dynamic conflict resolution between aircraft which is a significant improvement over the heuristic approaches commonly found in the industry. A CV module was developed to monitor controlled areas such as the runway, and an NLP pipeline was implemented for both intent recognition and flight entity identification, allowing for the possibility of integrating pilot communication into an autonomous system.

Although AI is scarce in operational ATC, this work has demonstrated the feasibility of applying DL to autonomous ground control. With further development and refinement, many of the components presented here could be extended to support human controllers in broader aviation contexts. While several workarounds were possible due to the simulated nature of this project, the challenges encountered also highlight the practical and theoretical gaps that need to be addressed before such systems can be considered for real-world deployment.

10.2 Limitations and Implications

Throughout this project, several challenges were encountered that reflect the current barriers to the adoption of AI in aviation. A notable case was the integration of MARL coordination between aircraft. Despite extensive experimentation and following best practices, the models consistently failed to learn stable or generalisable policies. The architecture itself appeared theoretically sound, and consultation with researchers in the field confirmed that the implementation strategy was valid, but the underlying cause of failure could not be identified.

This experience highlights one of the key limitations of current AI architectures, which is the lack of interpretability. As discussed in Section 2.5, the field of explainable AI (XAI) remains underdeveloped, and being unable to trace or understand the reasoning of models makes debugging extremely difficult. In contrast to traditional rule-based systems, DL models often require complete retraining or architectural redesign when performance issues arise. Moreover,

evaluation methods and metrics for such models are often task-specific and non-standardised, making it difficult to assess their effectiveness in a consistent or meaningful way across different scenarios [26].

Another significant challenge relates to data availability and generalisation. The LSTM and YOLO models developed in this project both demonstrated strong performance in the training environment but are unable to generalise to unseen scenarios. In practice, this would limit their applications in operational settings, especially in an industry where reliability and safety are held above everything. Moreover, since many AI-based aviation systems are protected and closed-source, it becomes difficult to verify performance claims or reproduce results.

Until AI systems can be made more transparent and robust across a wide range of conditions, their deployment in safety-critical domains such as ATC will stay limited.

10.3 Ethical, Legal and Safety Considerations

This project was conducted entirely within a simulated environment, using fully synthetic data generated from scratch. As such, there were no ethical concerns related to personal data, consent, or privacy. Nevertheless, future applications of AI in ATC involving real pilot communications or surveillance footage would fall under data protection laws such as the General Data Protection Regulation (GDPR) [77]. In such cases, careful attention would be required to ensure data is anonymised, securely stored, and processed lawfully in accordance with Article 6 of the GDPR.

From a legal standpoint, while this work remains a proof of concept, it touches on domains that are subject to stringent certification in real-world deployment. For example, frameworks such as DO-178C (Software Considerations in Airborne Systems) [33] and ARP4754A (System Development Assurance) [34] govern software safety in aviation. The absence of regulations for adaptive or learning systems currently limits the operational use of AI in ATC [25].

Ethically, the system was designed with safety and reliability as the main priorities. However, this project also highlighted common issues in AI systems, such as poor generalisation, lack of interpretability, and inconsistent performance under rare or edge-case scenarios. These challenges reinforce the importance of explainable AI, fairness in model behaviour, and the importance of keeping humans in the field.

Ultimately, this project contributes to the growing body of research exploring how AI might be responsibly and safely integrated into air traffic management.

10.4 Future Work

While this project successfully demonstrated a proof of concept for autonomous ground control using DL, there are several routes open for future work. A key area is the development of more robust and generalisable coordination mechanisms. Despite extensive experimentation, MARL failed to produce consistent or scalable policies. Future efforts may benefit from integrating

imitation learning, curriculum learning, or hybrid reward structures to improve coordination under decentralised conditions [78].

Another important limitation was the lack of generalisability across different airport layouts. The LSTM-based trajectory predictor and the YOLO model were both trained exclusively on data from Birmingham Airport, limiting their adaptability. Expanding the dataset to include multiple airport configurations would allow for a transferable system.

To move closer to real-world applications, future iterations should also explore compliance with aviation-specific standards such as DO-178C and ARP4754A and include explainability tools when making decisions. There is also large potential to integrate and normalise real-time speech input and extend the NLP component into a live communication interface between human pilots and AI systems. Arguable, the most important thing to note is; the human-in-the-loop framework where autonomous agents assist rather than replace controllers provides the most realistic and certifiable foundation for future air traffic systems.



11 References

- [1] European Parliament and Council, “Regulation (EC) No. 549/2004,” March 2004. [Online]. Available: <https://eur-lex.europa.eu/eli/reg/2004/549/oj/eng>. [Accessed 2025].
- [2] Skybrary Aviation Safety, “How air traffic control works,” [Online]. Available: <https://skybrary.aero/articles/how-air-traffic-control-works>. [Accessed 2025].
- [3] H. J. Older and B. J. Cameron, “Human Factors Aspects of Air Traffic Control,” National Aeronautics and Space Administration, 1972.
- [4] L. Hui, Z. Pei, S. Quan, X. Ke and S. Zhe, “Cognitive Workload Detection of Air Traffic Controllers Based on mRMR and Fewer EEG Channels,” *Frontiers in Psychology*, vol. 14, no. 8, p. 811, 2024.
- [5] K. J. Ruskin, S. Rice and C. A. Ruskin, “A Handbook for Signal Design: Alarms, Alerts, and Warnings in Air Traffic Control,” Federal Aviation Administration, 2022.
- [6] B. Hilburn, “Cognitive Complexity in Air Traffic Control – A Literature Review,” EUROCONTROL Experimental Centre, 2004.
- [7] J. Garcia, D.-B. Vo, A. Brock, V. Peyruqueou, A. Battut, M. Cousy, V. Canadyova, A. Sharpanskykh and G. Ermis, “Towards More Automated Airport Ground Operations Including Engine-Off Taxiing Techniques Within the Auto-Steer Taxi at AIRport (ASTAIR) Project,” *Engineering Proceedings*, vol. 90, no. 1, p. 15, 2025.
- [8] K. R. Dismukes, A. B. Berman and L. Loukopoulos, *The Limits of Expertise: Rethinking Pilot Error and the Causes of Airline Accidents*, Ashgate, 2007.
- [9] Japan Meteorological Agency, “Weather Phenomena Affecting Air Traffic Management Operations,” International Civil Aviation Organization (ICAO), 2015.
- [10] T. B. Spance, R. O. Fanjoy, C.-t. Lu and S. W. Schreckengast, “International standardization compliance in aviation,” *Journal of Air Transport Management*, vol. 49, pp. 1-8, 2015.
- [11] A. Hale, D. Borys and M. Adams, “Regulation or criminalisation: What determines legal standards of culpability in cases of workplace fatalities?,” *Safety Science*, vol. 113, pp. 96-107, 2019.
- [12] O. Omosebi, M. Azimi, D. Olowokere, Y. Wanyan, Q. Zhao and Y. Qi, “Investigating Runway Incursion Incidents at United States Airports,” *Future Transportation*, vol. 3, no. 4, pp. 1209-1222, 2023.
- [13] Japan Transport Safety Board, “Interim Report on the Aircraft Collision at Haneda Airport,” Japan Transport Safety Board, 2024.
- [14] National Transportation Safety Board, “Deconflict Airplane and Helicopter Traffic in the Vicinity of Ronald Reagan Washington National Airport,” National Transportation Safety Board, 2025.
- [15] G. Gui, Z. Zhou, J. Wang, F. Liu and J. Sun, “Machine Learning Aided Air Traffic Flow Analysis Based on Aviation Big Data,” *IEEE Transactions on Vehicular Technology*, vol. 69, no. 5, pp. 4817-4826, 2020.
- [16] E. Perl, “Review of Airport Surface Movement Radar Technology,” *IEEE Aerospace and Electronic Systems Magazine*, vol. 21, no. 10, pp. 24-27, 2006.

- [17] B. Tan, E. Lohan, B. Sun, W. Wang, T. Yesilyurt, C. Morlaas, C. Pena, K. Abdo, F. Slama, A. Simonin and M. Ellejmi, "Improved Sensing and Positioning via 5G and mmWave radar for Airport Surveillance," 2022.
- [18] K. J. Ruskin, C. Corvin, S. Rice, G. Richards, S. R. Winter and A. C. Ruskin, "Alarms, alerts, and warnings in air traffic control: An analysis of reports from the Aviation Safety Reporting System," *Transportation Research Interdisciplinary Perspectives*, vol. 12, 2021.
- [19] J. K. Kuchar and A. C. Drumm, "The Traffic Alert and Collision Avoidance System," *Lincoln Laboratory Journal*, vol. 16, no. 2, pp. 277-296, 2007.
- [20] Y. Wang, G. Xiao and Z. Dai, "Integrated Display and Simulation for Automatic Dependent Surveillance–Broadcast and Traffic Collision Avoidance System Data Fusion," *Sensors*, vol. 17, no. 11, p. 2611, 2017.
- [21] L. Z. Jun, S. Alam, I. Dhief and M. Schultz, "Towards a greener Extended-Arrival Manager in air traffic control: A heuristic approach for dynamic speed control using machine-learned delay prediction model," *Journal of Air Transport Management*, vol. 103, 2022.
- [22] A. Bikir, O. Idrissi, K. Mansouri and M. Qbadou, "Towards strengthening the efficiency of the Air Traffic Departure Manager DMAN using a Genetic Algorithm," in *International Conference on Wireless Networks and Mobile Communications (WINCOM)*, 2023.
- [23] International Civil Aviation Organization, "Air Traffic Management Automation System Implementation and Guidance Document," ICAO Asia and Pacific Office, 2022.
- [24] S. Mondoloni and N. Rozen, "Aircraft trajectory prediction and synchronization for air traffic management applications," *Progress in Aerospace Sciences*, vol. 119, 2020.
- [25] Y. Xie, N. Pongsakornsathien, A. Gardi and R. Sabatini, "Explanation of Machine-Learning Solutions in Air-Traffic Management," *Aerospace*, vol. 8, no. 8, p. 224, 2021.
- [26] M. Cocchioni, S. Bonelli, C. Westin, C. Borst, M. Bang and B. Hilburn, "Learning for Air Traffic Management: guidelines for future AI systems," *Journal of Physics: Conference Series*, vol. 2526, no. 1, 2023.
- [27] EUROCONTROL, "Real safety and capacity gains at Heathrow from artificial intelligence initiatives," 2020. [Online]. Available: <https://www.eurocontrol.int/article/real-safety-and-capacity-gains-heathrow-artificial-intelligence-initiatives>. [Accessed 2025].
- [28] . K. Yamazaki, S. Inoue, H. Aoyama and T. Kimiaki, "Remote Digital Tower to support Air Traffic Control Systems," in *Advances in Human Factors of Transportation*, 2023.
- [29] J. Zuluaga-Gomez, K. Vesely, I. Szoke, A. Blatt, P. Motliceck, M. Kocour, M. Rigault, K. Choukri, A. Prasad, S. Sarfjoo, I. Nigmatulina, C. Cevenini, P. Kolcarek, A. Tart, J. Cernocky and KI, "ATCO2 corpus: A Large-Scale Dataset for Research on Automatic Speech Recognition and Natural Language Understanding of Air Traffic Control Communications," 2023.
- [30] M. Kleinert, O. Ohneiser, H. Helmke, S. Shetty, H. Ehr, M. Maier, S. Schacht and H. Wiese, "Safety Aspects of Supporting Apron Controllers with Automatic Speech Recognition and Understanding Integrated into an Advanced Surface Movement Guidance and Control System," *Aerospace*, vol. 10, no. 7, p. 596, 2023.
- [31] S. Ghosh, S. Laguna, S. H. Lim, L. Wynter and H. Poonawala, "A Deep Ensemble Multi-Agent Reinforcement Learning Approach for Air Traffic Control," 2020.

- [32] G. Vouros, G. Papadopoulos, A. Bastas, J. M. Cordero and R. R. Rodrigez, “Automating the resolution of flight conflicts: Deep reinforcement learning in service of air traffic controllers,” 2022.
- [33] Radio Technical Commission for Aeronautics, “DO-178,” [Online]. Available: <https://www.rapitasystems.com/do178>. [Accessed 2025].
- [34] SAE International, “Guidelines for Development of Civil Aircraft and Systems,” [Online]. Available: <https://www.sae.org/standards/content/arp4754a/>. [Accessed 2025].
- [35] R. D.-A. Jurado, V. O. Plaza, X. Ye, M. Z. Suárez, F. P. Moreno and C. G. Arnaldo, “Establishing Rigorous Certification Standards: A Systematic Methodology for AI Safety-Critical Systems in Military Aviation,” *IEEE Access*, vol. 12, 2024.
- [36] C. H. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk and Brett, “Array programming with NumPy,” *Nature*, vol. 585, pp. 357-362, 2020.
- [37] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bardbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani and Chila, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” *Advances in Neural Information Processing Systems*, 2019.
- [38] “OpenCV,” [Online]. Available: <https://opencv.org/>. [Accessed 2025].
- [39] “Ultralytics,” [Online]. Available: <https://www.ultralytics.com/>. [Accessed 2025].
- [40] “Pandas,” [Online]. Available: <https://pandas.pydata.org/>. [Accessed 2025].
- [41] “Scikit-learn: Machine Learning in Python,” [Online]. Available: <https://scikit-learn.org/>. [Accessed 2025].
- [42] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. v. Platen, C. Ma, Y. Jernite, J. Plu and C. Xu, “HuggingFace’s Transformers: State-of-the-art Natural Language Processing,” *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38-45, 2020.
- [43] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” in *International Conference for Learning Representations*, San Diego, 2015.
- [44] Pygame Community, “Pygame Documentation,” [Online]. Available: <https://www.pygame.org/docs/>. [Accessed 2025].
- [45] Wikipedia contributors, “Birmingham Airport,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Birmingham_Airport. [Accessed 2025].
- [46] ICAO, “EGBB: Birmingham Airport,” OpenNav, [Online]. Available: <https://opennav.com/airport/EGBB>. [Accessed 2025].
- [47] Boeing, “Boeing Next-Generation 737,” [Online]. Available: <https://www.boeing.com/commercial/737ng>. [Accessed 2025].
- [48] E. W. Dijkstra, “A note on two problems in connexion with graphs,” in *Numerische Mathematik*, 1959, pp. 269-271.

- [49] P. E. Hart, N. J. Nilsson and R. Bertram, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100-107, 1968.
- [50] A. M. Lacorte and E. P. Chavez, "Analysis on the Use of A* and Dijkstra's Algorithms for Intelligent School Transport Route Optimization System," in *Proceedings of the 4th International Conference on Human-Computer Interaction and User Experience in Indonesia (CHIUXID '18)*, 2018.
- [51] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, vol. 2, MIT Press, 2018.
- [52] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou and H. King, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529-533, 2015.
- [53] Y. Li, "Deep Reinforcement Learning: An Overview," *arXiv preprint arXiv:1701.07274*, 2017.
- [54] U.S. Federal Aviation Administration, "Aeronautical Information Manual (AIM)," 2017. [Online]. Available: https://www.faa.gov/air_traffic/publications/media/aim.pdf. [Accessed 2025].
- [55] Y. Flet-Berliac and P. Preux, "MERL: Multi-Head Reinforcement Learning," *arXiv preprint arXiv:1909.11939*, 2019.
- [56] L. Busoniu, R. Babuska and B. De Schutter, "A Comprehensive Survey of Multiagent Reinforcement Learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, vol. 28, no. 2, pp. 156-172, 2008.
- [57] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel and I. Mordatch, "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments," in *31st Conference on Neural Information Processing Systems (NIPS)*, Long Beach, CA, 2017.
- [58] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 8, no. 8, pp. 1735-1780, 1997.
- [59] S. Lee, Y. Seo, K. Lee, P. Abbeel and J. Shin, "Offline-to-Online Reinforcement Learning via Balanced Replay and Pessimistic Q-Ensemble," *arXiv preprint arXiv:2107.00591*, 2021.
- [60] T. Schaul, J. Quan, I. Antonoglou and D. Silver, "Prioritized Experience Replay," in *4th International Conference on Learning Representations*, San Juan, 2016.
- [61] S. Kalyanakrishnan, S. Aravindan, V. Bagdawat, V. Bhatt, H. Goka, A. Gupta, K. Krishna and V. Piratla, "An Analysis of Frame-skipping in Reinforcement Learning," *arXiv preprint arXiv:2102.03718*, 2021.
- [62] W. Fedus, P. Ramachandran, R. Agarwal, Y. Bengio, H. Larochelle, M. Rowland and W. Dabney, "Revisiting Fundamentals of Experience Replay," in *3th International Conference on Machine Learning (ICML)*, 2020.
- [63] D. K. S. Nandini and S. V. R., "Comparative Study of Machine Learning Algorithms in Detecting Cardiovascular Diseases," *arXiv preprint arXiv:2405.17059*, 2024.
- [64] ICAO, "ICAO Doc 9157 Aerodrome Design Manual Part 2," 2013. [Online]. Available: <https://skybrary.aero/bookshelf/icao-doc-9157-aerodrome-design-manual-part-2>. [Accessed 2025].
- [65] S. Pateria, B. Subagdja, A.-h. Tan and C. Queck, "Hierarchical Reinforcement Learning: A Comprehensive Survey," *ACM Computing Surveys*, vol. 54, no. 5, pp. 1-35, 2021.

- [66] O. E. Olorunshola, A. K. Ademuwagun and C. Dyaji, “Comparative Study of Some Deep Learning Object Detection Algorithms: R-CNN, FAST RCNN, FASTER R-CNN, SSD, and YOLO,” *Nile Journal of Engineering and Applied Science*, vol. 1, no. 1, pp. 70-80, 2023.
- [67] A. Radford, K. Narasimhan, T. Salimans and I. Sutskever, “Improving Language Understanding by Generative Pre-Training,” OpenAI, 2018.
- [68] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.
- [69] Homophone.com, “Homophone Dictionary,” [Online]. Available: <https://www.homophone.com/>. [Accessed 2025].
- [70] Datamuse, “Datamuse API,” [Online]. Available: <https://www.datamuse.com/api/>.
- [71] J. Wei and K. Zou, “EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks,” in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Hong Kong, 2019.
- [72] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions and reversals,” *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707-710, 1966.
- [73] V. Sanh, L. Debut, J. Chaumond and T. Wolf, “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter,” arXiv preprint arXiv:1910.01108, 2019.
- [74] R. Nogueira and K. Cho, “Passage Re-ranking with BERT,” arXiv preprint arXiv:1901.04085, 2019.
- [75] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland and T, “Manifesto for Agile Software Development,” 2001. [Online]. Available: <https://agilemanifesto.org/>. [Accessed 2025].
- [76] W. Jiang, J. Luo, M. He and W. Gu, “Graph Neural Network for Traffic Forecasting: The Research Progress,” *ISPRS International Journal of Geo-Information*, vol. 12, no. 3, p. 100, 2023.
- [77] “Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data (General Data Protection Regulation),” *Official Journal of the European Union*, pp. 1-88, 2016.
- [78] D. Blessing, O. Celik, X. Jia, M. Reuss, M. X. Li, R. Lioutikov and G. Neumann, “Information Maximizing Curriculum: A Curriculum-Based Approach for Imitating Diverse Skills,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

12 Appendix



Figure 12.A: Aircraft sprite development from first iteration to final sprite.

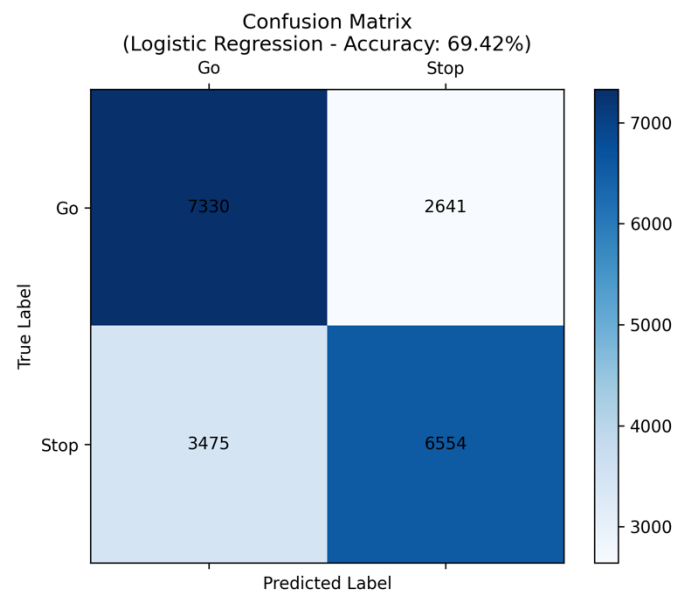


Figure 12.B: Logistic Regression confusion Matrix on Collision Avoidance Dataset



Figure 12.C: Failure of LSTM model to predict go around positions due to the unseen scenario.

Results of the practical experiment done with the NLP model described in Section 7.6. Green cells highlight correct predictions, and red cells highlight incorrect predictions.

Actual Request	Transcribed Request
Iberia one six seven holding at Alpha Two, request departure clearance and wind direction confirmation.	Iberia 167 holding at Alpha two request departure clearance and wind direction confirmation
Singapore six eight eight, brakes set and engines stable, request takeoff clearance from active runway.	Singapore 688 brick set in England stable request takeoff clearance from active runway
Tower, Singapore six seven five, all systems go, request takeoff from runway Three Three Right.	tower Singapore 675 all systems go request tax off from runway 33 right
Tower, this is American three three zero, fully configured for landing, request immediate clearance to touch down.	this is the American 330 fully configured for landing request immediate clearance to touchdown
Ground, Cathay six five four, ready for taxi, requesting taxiway routing to departure runway.	grounds 654 ready for taxi requesting taxi we rooting to watch your runway

Table 12.A: Test User 1 – 43.4% Word Error Rate in Transcription.

Actual Request	Transcribed Request
United four five two holding short of runway, ready for immediate departure, requesting takeoff clearance.	United 452 holding short of runway ready for immediate departure requesting takeoff clearance
Tower, this is Lufthansa nine five eight, fully configured for landing, request immediate clearance to touch down.	tell her this is Lathan 958 full effect for landing requested clearance of touchdown
Ground control, Iberia one seven five at terminal gate requesting permission to commence pushback for scheduled departure.	ground control Iberia 175 at terminal gate requesting permission to commence pushback for scheduled departure
Ground, American six seven eight, fully boarded and ready at gate, requesting pushback followed by taxi instructions.	ground American 678 fully boarded and ready at gate requesting pushback followed by taxi instructions
Lufthansa seven four two requesting pushback clearance from stand Four Two, systems all checked and ready.	Lufthansa 742 requesting pushback clearance from stand for two systems all check then ready

Table 12.B: Test User 2 – 34.1% Word Error Rate in Transcription.

Actual Request	Transcribed Request
Qantas four four seven, brakes set and engines stable, request takeoff clearance from active runway.	Qantas 447 brick set engine stable request take clearance from active runway
This is KLM two one one on stand One Zero requesting pushback and engine start for departure to Dubai.	this is KLM 211 on Stan 10 requesting pushback and engine start for departure to Dubai
Turkish nine three one on approach to runway One Eight, requesting clearance and confirmation of traffic.	Turkish 931 on approach to runway 1/8 requesting clearance and confirmation of traffic
Ground, Southwest six zero two, fully boarded and ready at gate, requesting pushback followed by taxi instructions.	ground Southwest 620 fully Boyden ready at the gate requesting pushback followed by taxi instructions
Tower, Lufthansa three one seven, request taxi clearance to runway Zero Six Right, we're ready and holding.	tower Lufthansa 317 request taxi clearance to runway 06 Wright we're ready and holding

Table 12.C: Test User 3 – 35.3% Word Error Rate in Transcription.

Actual Request	Transcribed Request
United seven six nine on approach to runway One Eight, requesting clearance and confirmation of traffic.	Unit 769 on approach to runway one a question clearance and confirmation of traffic
Qantas eight seven one, brakes released and steering engaged, request taxi route to holding point Alpha Four.	auto 871 Reese released at steering engaged request taxi route to Alpha
Ground, Delta two zero three, ready for taxi, requesting taxiway routing to departure runway.	ground Delta 203 ready for taxi requesting taxi we rooting to departure runway
Delta six one two holding short of runway, ready for immediate departure, requesting takeoff clearance.	Delta 612 Holden short of ready for immediate departure requesting takeoff clearance
Ground, this is Air France five one three, requesting clearance to taxi to runway via standard routing.	this is 513 requesting clearance taxi to runway via under rooting

Table 12.D: Test User 4 – 44.3% Word Error Rate in Transcription.

Actual Request	Transcribed Request
All Nippon four two seven to ground, pushback complete and brakes released, requesting taxi clearance.	1.4272 ground pushback complete and brakes release requesting taxi clearance
Cathay five three three, brakes set and engines stable, request takeoff clearance from active runway.	café 533 brake set and engine stable request takeoff clearance from active runway
Alitalia three three six at parking bay Two Five requesting permission for pushback, engines to be started during movement.	Italia 336 @parking bay 25 requesting permission for pushback engines to be started during movement
American one eight one on approach to runway One Eight, requesting clearance and confirmation of traffic.	American 181 on approach to runway 1/8 requesting clearance and confirmation of traffic
This is Southwest five six two, aligned on runway One Eight, requesting clearance for takeoff.	this is Southwest 562 aligned on runway 1/8 requesting clearance for takeoff

Table 12.E: Test User 5 – 37.5% Word Error Rate in Transcription.