2025-01-31

Engineering Design Automation via Imitation Learning and Reinforcement Learning

Bozorgmehry Boozarjomehry, Ghazal

UNIVERSITY OF CALGARY

Engineering Design Automation via Imitation Learning and Reinforcement Learning

by

Ghazal Bozorgmehry Boozarjomehry

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

GRADUATE PROGRAM IN MECHANICAL ENGINEERING

CALGARY, ALBERTA

JANUARY, 2025

Abstract

Reinforcement Learning (RL) has achieved notable success in robotics and gaming, yet its application to automating engineering design faces significant challenges, including slow training times and poor generalization. Traditional RL methods require exploring millions of design states, which is computationally expensive, especially when dealing with complex physics models. In contrast, behavioral cloning, which enables RL agents to mimic human designers based on their decision data, presents a more resource-efficient alternative. This thesis investigates the use of both RL and imitation learning to automate engineering design, using aircraft design as a surrogate task to model engineering design.

We evaluate the performance of a behavioral cloning agent trained on human design decision data, employing recurrent neural networks such as GRU, LSTM, and simple-RNN. We define a metric Q-score, which quantifies design quality on a scale between 0 and 1, with higher values indicating better design quality. Our findings demonstrate that the GRU architecture outperforms both LSTM and simple-RNN in terms of accuracy, achieving a Q-score of 0.8 after training on a relatively small dataset. The GRU model strikes an optimal balance between accuracy, simplicity, and computational efficiency, making it particularly suitable for surrogate design tasks like aircraft design optimization.

Additionally, we assess the performance of RL agents, specifically Proximal Policy Optimization and Advantage Actor-Critic, in the same design task. Both RL approaches achieved higher Q-scores (up to 0.99) but incurred significant computational costs and required extensive training time. In contrast, behavioral cloning provided a faster, more computationally efficient approach, though its performance was constrained by the availability of labeled human decision data.

The results suggest that while RL methods excel in exploration and optimization, imitation learning offers a faster and more resource-efficient solution, albeit with reduced exploration and adaptability.

To My Beloved Family

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Engineering design is a critical component of modern society, serving as the foundation for nearly all technological advancements. It plays a vital role in creating the infrastructure that provides shelter, vehicles that enable global connectivity, and consumer products that enhance our daily lives. This multidisciplinary field combines creativity and technical expertise to transform concepts into tangible solutions.

In recent years, the emergence of artificial intelligence (AI) and machine learning (ML) has transformed numerous industries by automating complex tasks and enabling unprecedented levels of efficiency and accuracy. These technologies hold significant potential to revolutionize the engineering design process as well, offering the possibility to enhance creativity, optimize designs, and reduce development cycles. For example, AI-driven algorithms can analyze vast datasets to identify patterns and generate innovative solutions that might elude human designers. However, despite the promise of AI and ML, much of the engineering design workflow remains reliant on manual input and traditional methodologies.

The slow adoption of AI and ML in engineering design can be attributed to several challenges. One of the primary obstacles is the reliance on computationally expensive physics-based models, which are essential for simulating and validating designs. These models often require significant resources, making them impractical for iterative or real-time applications. Another major challenge is the difficulty in generalizing AI and ML algorithms across multiple design tasks. Unlike narrowly defined applications, engineering

design encompasses a diverse range of problems, each with unique requirements and constraints. This diversity limits the transferability of machine learning models and increases the demand for task-specific data, which is often scarce or costly to obtain.

By overcoming these barriers, engineering design processes could become more accessible, scalable, and adaptive to the ever-changing demands of modern society. This thesis aims to explore these challenges and propose strategies to bridge the gap between traditional engineering design practices and the opportunities presented by AI and ML technologies. Specifically, the focus of this thesis is on the first obstacle. Computationally expensive physics-based models pose significant obstacles to data-hungry ML algorithms. Innovations in model efficiency could pave the way for more widespread automation and significantly impact the field of engineering design automation.

1.1 Engineering Design as Markov Decision Process

Engineering design is fundamentally a process of sequential decision-making, wherein an agent interacts with a design environment to create a solution that meets a predefined set of requirements [Miller et al. (2018); Boozarjomehry and Thekinen (2023)]. This process involves iterative evaluation and optimization, requiring the agent to navigate complex trade-offs among competing objectives. Engineering design integrates a wide range of knowledge, including mathematical models, empirical research, and design heuristics, to develop solutions that fulfill desired requirements within specific performance parameters.

In this thesis, this sequential engineering design decision-making process is modeled using a Markov Decision Process (MDP). As shown in Figure 1.1, this framework allows engineers to systematically explore and optimize design spaces as a sequential decision-making process. Each state represents a specific design configuration, actions correspond to modifications made to the design, rewards measure the performance of these modifications, and transitions capture the likelihood of moving from one state to another. The MDP formulation is described in depth in Chapters 2 and 3.

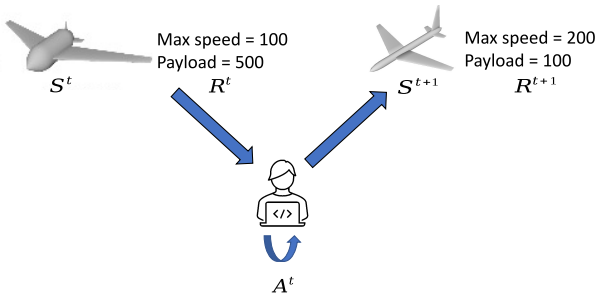


Figure 1.1: Engineering design as a Markov decision process.

In theory, a policy trained via reinforcement learning can automatically solve engineering design MDP. The reinforcement learning approach is a branch of machine learning that focuses on training agents to make decisions sequentially, aiming to maximize cumulative rewards in a specified environment. The principle behind this technique is to make the agents learn to perform actions that yield the highest rewards while adapting their strategies to the feedback they receive from the environment. The potential of RL has been demonstrated in diverse applications, including robotics [Zhao et al. (2020)], autonomous driving [Sallab et al. (2017)], energy systems [Perera and Kamalaruban (2021)], and complex strategy games [Silver et al. (2018)]. In some tasks requiring high-precision data processing and decision-making, RL agents even outperform humans, provided that they have thoroughly searched the state space and used appropriate reward mechanisms [Zhao et al. (2020),Arulkumaran et al. (2017)]. These successes highlight RL's capacity to handle dynamic and high-dimensional problems, making it a natural fit for engineering design tasks. Therefore, an effectively trained RL agent should be capable of achieving human-level performance in specific engineering design tasks, such as trade space exploration and parametric design, where particular variables must be optimized within predefined constraints.

1.2 Challenge with RL

Despite the theoretical potential, the practical application of RL to engineering design faces significant barriers. Training an RL agent from scratch typically requires millions of

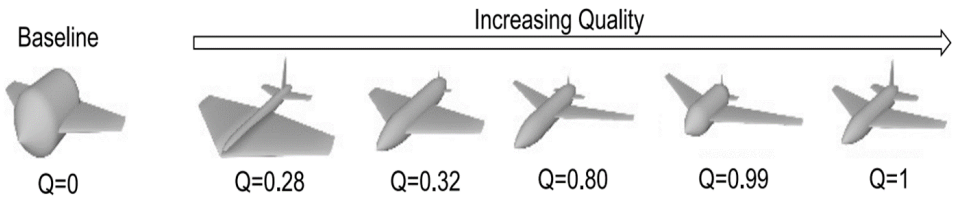


Figure 1.2: Q-score of various designs produced by human designers.

iterations, as evidenced by landmark successes in areas like AlphaGo [Silver et al. (2017)]. This training process often demands substantial computational resources, which is a major limitation in engineering design contexts. For example, a single stress or performance analysis in an engineering simulation can take several hours to compute. When multiplied across the thousands of episodes required for RL training, the computational expense becomes prohibitively high. This challenge underscores the need for innovative approaches to adapt RL methods for resource-intensive domains such as engineering design.

1.3 Motivation Behind Key Idea

We observe that human designers exhibit remarkable efficiency and adaptability when solving new design tasks, often achieving satisfactory results with relatively few iterations. Unlike RL agents, which typically require extensive training and optimization, humans can leverage prior experience and domain knowledge to quickly explore and refine design solutions. For example, a prior human subject study on engineering students [Thekinen and Grogan (2021, 2022)] demonstrated this efficiency in an aircraft design task. Figure 1.2 illustrates five aircraft designs and their corresponding quality scores (Q-scores) from that study. The Q-score is a numerical measure of design quality, ranging from 0 to 1, where a score of 1 indicates that the design has fully satisfied all required criteria. In this study, novice designers started from a baseline design, which had a Q-score of 0, and, on average, required fewer than 100 actions to produce high-quality design results.

In stark contrast, curiosity-driven RL algorithms tailored to sparse-reward settings [Pathak

et al. (2017)] demand thousands of actions to achieve comparable Q-scores. One of the key strengths of human designers is their ability to generalize knowledge and build robust mental models that can be applied across diverse problems. By drawing on prior experiences, humans are able to prioritize meaningful actions, navigate complex design spaces, and adapt quickly to new challenges. In contrast, RL agents are fundamentally constrained by their reliance on large quantities of task-specific data and their limited capacity to transfer learned behaviors to novel tasks. As a result, human designers and deep RL agents are effectively solving different problems, each governed by distinct mechanisms of reasoning and decision-making.

While RL agents excel in tasks requiring brute-force exploration and optimization, their inefficiency in sparse-reward environments and inability to generalize across diverse problems limit their practicality in design contexts. In contrast, human designers can efficiently navigate complex design spaces by drawing on prior knowledge and applying heuristics, often achieving high-quality results with minimal iterations.

One promising approach to addressing this challenge is the integration of human expertise with RL algorithms. By transferring human-derived heuristics to RL agents, it is possible to enhance their performance in tasks requiring sequential decision-making. Such methods have already shown success in specific domains, such as video gaming, where human guidance has been used to improve sample efficiency and accelerate learning [Dubey et al. (2018)]. This approach not only leverages the efficiency and intuition of human problem-solving but also retains the flexibility and scalability of RL systems.

1.4 Research Gap and Questions

Despite its potential, the application of human-influenced RL methods to engineering design remains largely unexplored. Currently, there are no established datasets or algorithms specifically designed to train RL agents for engineering activities such as trade space exploration. Trade space exploration, a critical component of engineering design, involves evaluating and optimizing multiple design alternatives to identify the best solution under

given constraints. Automating this process with RL could revolutionize modern engineering design by significantly reducing manual effort, accelerating development cycles, and enabling designers to focus on more creative and strategic aspects of the design process.

Developing methods that combine human heuristics with RL for engineering design could pave the way for transformative advancements in the field. These methods have the potential to improve sample efficiency, shorten training times, and make RL more applicable to real-world design problems.

Transferring human decision-making expertise to RL agents is a promising strategy for improving their performance in engineering design tasks. An effective approach to achieve this transfer is through imitation learning. Imitation learning is a machine learning paradigm that allows RL agents to acquire complex behaviors by mimicking the actions of a human expert [Hussein et al. (2017)]. By observing and replicating expert demonstrations, RL agents can learn to navigate decision-making processes that would otherwise require extensive trial-and-error training. This method has gained considerable attention in recent years, with successful applications in domains such as robotics, autonomous driving, and gaming.

Several methods of imitation learning have been developed to transfer human reward functions to RL agents. These include behavioral cloning, learning from demonstration [Argall et al. (2009)], and generative adversarial imitation learning [Ho and Ermon (2016a)]. Among these, behavioral cloning is one of the simplest and most widely used approaches. It involves training an agent to replicate the actions of a human expert by mapping observed inputs to corresponding actions [Torabi et al. (2018)].

Despite the promise of behavioral cloning in various domains, its effectiveness in replicating human decision-making specifically in engineering design remains unexplored. While behavioral cloning has demonstrated success in areas such as robotics and autonomous systems, there is little research investigating its application to decision-making processes of human designers in engineering contexts. Furthermore, no existing studies systematically compare different behavioral cloning architectures in their ability to predict human design decisions.

This thesis aims to fill this research gap. Specifically, we investigate the ability of these policies to automatically generate designs that meet predefined requirements, starting from a baseline design. By doing so, we aim to determine how well behavioral cloning can replicate the strategies and decisions of human designers in a complex, iterative design environment.

This thesis address the following research questions:

- What is the accuracy of a behavioral cloning policy in predicting human decisions in engineering design? This question aims at providing insights into potential use of behavioral cloning as a tool for automating design tasks.
- How does the accuracy of various sequence-learning-based behavioral cloning architectures compare in predicting human design decision-making?
- How does the performance of imitation learning methodologies compare to reinforcement learning approaches in the context of engineering design applications?
- How does the sample efficiency of imitation learning compare to that of reinforcement learning for engineering design tasks?

By answering these research questions, this thesis seeks to contribute to the growing body of knowledge on applying machine learning to engineering design. The findings are expected to provide a foundation for future work in integrating human-inspired decision-making processes into automated design frameworks.

To investigate the potential of behavioral cloning in replicating human decision-making in engineering design, we use data collected from a design experiment on a virtual design studio as the experimental environment. This virtual studio serves as a surrogate platform for conducting design tasks, enabling the systematic collection of data on human design decision-making. Specifically, our study focuses on aircraft design, a domain that involves complex, iterative decision-making processes. The experimental setup involves human designers starting from a baseline design and iteratively updating design parameters to meet pre-specified requirements. After each iteration, designers observe a reward function

that quantifies how well the updated design satisfies the given requirements. The data collected from these experiments form the basis for training and evaluating behavioral cloning models. We compare the performance of three sequence-learning-based behavioral cloning architectures to predict human design decisions: simple recurrent neural networks, long short-term memory networks, and gated recurrent units. These architectures are well-suited for modeling sequential data and capturing temporal dependencies, making them ideal for predicting iterative design decisions. By evaluating and comparing their accuracy, we aim to identify the most effective approach for replicating human decision-making patterns in the context of engineering design. The best-performing sequence-learning architecture is identified and further evaluated for its capabilities by forward simulating 250 design iterations, starting from the baseline design. This simulation assesses the ability of the behavioral cloning agent to autonomously generate a design that meets the pre-specified requirements. The results of this evaluation provide insights into the feasibility of using behavioral cloning to automate design.

In addition to evaluating imitation learning, we compare its performance against RL approaches. The comparison is based on two key metrics: the quality of the best design produced and the sample efficiency of the methods. To ensure a fair evaluation, we selected two RL algorithms known for their high sample efficiency: Proximal Policy Optimization and Advantage Actor-Critic. These algorithms are compared with imitation learning to assess their relative effectiveness in engineering design tasks.

1.5 Contributions

The main contributions of this thesis are follows:

- Modeling Tradespace Parameter Exploration as a Markov Decision Process: This thesis introduces a novel approach by modeling tradespace parameter exploration in the preliminary engineering design phase as a Markov decision process. Inspired by the sequential nature of engineering design decision-making, this work formulates design tasks as decision-making processes where each design iteration builds upon the

previous one. This perspective enables the application of advanced decision-making techniques, such as reinforcement learning and imitation learning, to optimize the exploration of design alternatives.

- Applying Imitation Learning to the Design MDP: This is the first study to apply imitation learning to the design MDP generated from human design decision data. By leveraging behavioral data from human designers, we demonstrate that imitation learning algorithms can effectively replicate human decision-making in the engineering design process. Furthermore, we show that imitation learning can achieve higher sample efficiency compared to traditional reinforcement learning techniques when solving the same design MDP, providing an efficient pathway for automating design tasks with fewer training iterations.

- Comparing Sequence Learning Architectures for Predicting Human Design Decisions: This thesis is the first to compare various sequence learning architectures in predicting human design decision-making. Specifically, we evaluate the performance of several sequence-based models, including simple recurrent neural networks, long short-term memory networks, and gated recurrent units.

The thesis is structured as follows: Chapter 2 provides the background, relevant literature, and formulation of the problem. Chapter 3 describes our approach to collecting human design decision data. Chapter 4 presents the behavioral cloning study using the design data. Chapter 5 explores the reinforcement learning study on the engineering design MDP. Chapter 6 concludes with a discussion of the findings and outlines directions for future work.

Chapter 2

Background and Problem Formulation

2.1 Engineering Design

Engineering design is a complex and dynamic field. It involves solving problems and creating innovative solutions using a structured problem-solving approach. Engineers blend technical analysis with creativity to fulfill specific functional requirements. This iterative process relies on feedback from various stakeholders to refine solutions, ensuring robustness and practicality in the final design. The interdisciplinary nature of engineering design integrates methodologies, computational tools, and collaborative efforts to achieve effective outcomes.

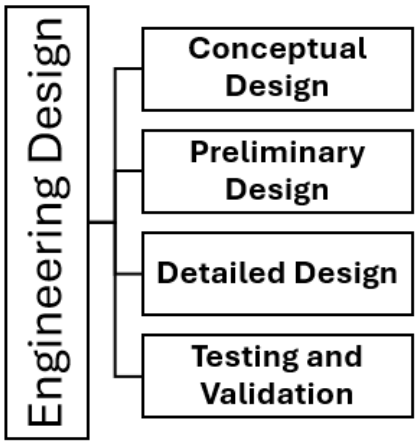


Figure 2.1: Engineering Design Steps

The design process is typically divided into four distinct phases as illustrated in Figure 2.1: conceptual design, preliminary design, detailed design, and testing and validation:

Conceptual Design: This phase focuses on exploring a broad range of possibilities to define the core functionalities of the product or system. Engineers brainstorm ideas, create sketches, and develop basic models to evaluate different concepts. The goal is to identify promising solutions that address the identified needs and constraints [AZARM (2009)].

Preliminary Design: In this phase, the concept chosen from the conceptual stage is refined into a more detailed plan. Engineers use tools such as computer-aided design software to create digital models and simulations, assessing feasibility, functionality, and performance. During this phase, material selection, basic specifications, and cost estimations are also established during this phase [Pahl et al. (2007)].

Detailed Design:This phase involves creating precise specifications and technical drawings for every component of the design. Engineers define dimensions, tolerances, materials, manufacturing processes, and assembly instructions to ensure that all aspects work together seamlessly [Ullman (2010)].

Testing and Validation: The final phase involves building prototypes and performing rigorous testing to ensure that the design performs as intended under real-world conditions. Engineers analyze results to identify any weaknesses or shortcomings through rigorous testing and refine the design as necessary before production.

The design process, with its distinct phases – conceptual design, preliminary design, detailed design, and testing and validation – lays the groundwork for informed decision-making. Each phase inform the critical choices engineers make throughout the design journey. However, the vast amount of information and numerous design possibilities, particularly in the early conceptual and preliminary design phases, can make informed decision-making a complex task. Here, engineers must grapple with uncertainty about various factors like material properties, manufacturing feasibility, and potential user needs [O'Connor and Kleyner (2012)]. While creativity is essential for generating new ideas, decision-making is the engine that drives effective engineering design. Every phase of the design process necessitates critical choices, from selecting the most promising concepts to defining material

specifications and manufacturing methods.

2.2 Challenges in Traditional Engineering Design Methods

Traditional design methodologies often rely on intuition, experience, and brainstorming. For example, manual drafting techniques were historically used to create initial designs, while rule-based design approaches, such as those in structural engineering, use predefined heuristics to develop solutions. While these methods have been fundamental in engineering, they pose challenges in efficiently exploring and evaluating the vast array of design possibilities.

The preliminary design phase is a critical juncture in the engineering design process. It acts as a bridge between the initial brainstorming of the conceptual design and the detailed technical specifications of the final design. Here, the focus shifts towards translating broad ideas into a more concrete form, while also considering practical constraints and feasibility. However, this phase is riddled with its own set of challenges that can significantly impact the success of the entire project. Some of the challenges are:

Balancing Creativity with Reality:The preliminary design phase demands a delicate balance between the creative freedom of the conceptual design and the harsh realities of manufacturability, cost, and material limitations. Overly ambitious designs from the conceptual phase can become impractical during preliminary design if they are not grounded in feasibility assessments. This can lead to significant revisions and delays later on [Pahl et al. (2007)].

Incomplete Information and Uncertainty: At this stage, complete information about the project, including final specifications or manufacturing details, may not be readily available. This lack of certainty can lead to design flaws or inefficiencies. Engineers rely on estimates and assumptions, which can later prove to be inaccurate and necessitate costly rework in the detailed design phase [Cross (2021)].

Managing Scope Creep: Preliminary design often sparks further discussions and clarifications with stakeholders. While this is a crucial step in refining the design, it can also

introduce new requirements or modifications. This "scope creep" can disrupt the planned design flow, increase complexity, and potentially delay project timelines. Managing stakeholder expectations and maintaining focus on core functionalities is essential [Kerzner (2017)].

Communication Gaps and Silos: Preliminary design often involves collaboration between different engineering disciplines. However, communication gaps between these teams can lead to inconsistencies or errors that are discovered later in the process. For instance, the mechanical design might not take into account limitations in the chosen material for manufacturing, identified by the materials engineer. These inconsistencies will require costly rework in the detailed design phase, highlighting the importance of clear and consistent communication throughout the process [Eppinger and Browning (2012)].

Traditional engineering design methodologies often struggle to address the inherent complexity of the preliminary design phase, particularly when dealing with incomplete information and the need to consider numerous, often conflicting, design objectives simultaneously. These limitations can lead to suboptimal design choices with significant downstream consequences. Engineering design automation aims to bridge this gap and facilitate more informed decision-making. By leveraging computational tools and mathematical frameworks, engineering design automation offers promising possibilities for streamlining and optimizing the preliminary design phase [Ullman (2010)].

2.3 Engineering Design Automation and Associated Challenges

In recent years, there has been a big push towards automating parts of the design process, driven by the need for speed, efficiency, and precision. Engineering design automation is a set of powerful tools and techniques that use the power of computers to automate repetitive tasks and support informed decision-making throughout the design process. These tools enable engineers to construct precise digital replicas of physical components and systems—often called 'digital twins'—which facilitate enhanced visualization, simulation,

and optimization throughout the design process [Ricondo et al. (2021), Lowe and Hartman (2011)].

Engineering design automation unlocks a multitude of benefits:

Faster Design Exploration: Automation tools like rule-based design can automatically generate variations based on pre-defined parameters, allowing engineers to explore a wider range of design possibilities quickly and efficiently [Chavali et al. (2008)].

Early Performance Insights: Finite Element Analysis (FEA) tools within EDA software can perform basic stress and strain analysis on designs during the early stages. This provides valuable insights into potential performance issues before physical prototypes are built, saving time and resources [Saadi and Yang (2023), Medyna et al. (2012)].

Streamlined Documentation: 3D models can automatically generate detailed and accurate engineering drawings, eliminating the need for manual drafting and reducing the risk of errors [Autodesk, Vaskỳ et al. (2010)].

However, despite these promising advantages, the path to fully realizing the potential of engineering design automation is fraught with challenges. Key among these is the difficulty in automating the design process itself, particularly due to the computational cost of high-fidelity physics models and the complexity of developing effective surrogate models. Addressing these challenges is essential for harnessing the full power of automation in engineering design. The two main challenges are described below:

1) Computational cost of physics model: These models, while offering highly accurate simulations, are computationally expensive to run, especially during design optimization involving numerous iterations. Running thousands of simulations to fine-tune a single design parameter requires substantial computational power and time, creating a significant bottleneck. This can slow down the entire design process and hinder its efficiency.

To mitigate these computational challenges, researchers often turn to surrogate models. These models aim to provide approximate solutions that are less computationally intensive than their high-fidelity counterparts. While surrogate models can significantly speed up the design process, developing these models introduces additional complexities. Creating accurate and reliable surrogate models requires specialized knowledge in selecting appro-

priate training data, validating the models, and balancing trade-offs between accuracy and computational efficiency. Additionally, surrogate models must be regularly updated and validated to reflect the evolving nature of design problems, adding another layer of difficulty to the automation process.

Despite these challenges, the effective incorporation of surrogate models into engineering workflows is a critical step toward streamlining design automation, albeit one that requires careful collaboration and clear communication across multidisciplinary teams. The integration of these advanced modeling techniques into engineering design automation also necessitates interdisciplinary collaboration and effective communication among various engineering teams. Ensuring that all team members have a clear understanding of the models and their limitations is crucial for maintaining consistency and avoiding errors that could lead to costly rework [Pestourie et al. (2023)].

2) Generalization over tasks: While machine learning, particularly deep learning, has revolutionized various aspects of engineering design automation, its full potential remains somewhat limited due to the challenge of generalizability. As highlighted by Liu et al. [Liu et al. (2018)], these algorithms often struggle to adapt their knowledge effectively when applied to new design tasks or domains. This is similar to the challenges faced in natural language processing [Socher et al. (2013)], where models trained for one specific task, such as sentiment analysis, might not translate well to another, like machine translation.

This limitation manifests in several ways within engineering design automation. For instance, a deep learning model trained to optimize the heat dissipation of a specific electronic component might not perform well when tasked with optimizing the structural integrity of a bridge. Similarly, training a model for selecting lightweight materials for aircraft wings would not necessarily translate to selecting materials for a building façade, which requires balancing factors like weight, aesthetics, and energy efficiency.

This lack of generalizability restricts the broader application of machine learning in design automation. Currently, each specific design task might require a dedicated model with its own training data. This creates a bottleneck for efficiency and limits the scalability of automation across different design domains.

The limitations of Engineering Design Automation methodologies highlight the need for new approaches and frameworks to support effective decision-making in engineering design. One potential approach is the use of Markov decision processes (MDP) to model decision-making procedures employed in preliminary stages of the design process. The next section describes formulation of engineering design as a Markov decision process, which is a key novelty of this thesis.

2.4 Engineering Design Formulated as Markov Decision Process

2.4.1 MDP Components in Engineering Design Context

An MDP is mathematically defined by the tuple $< S, A, O, T, R, \gamma >$. This section details how each component of the MDP tuple maps to the corresponding elements of the engineering design process.

State Space (S): S is the set of all possible states. In the context of engineering design, a state corresponds to a specific point in the design parameter space. State space encompasses all potential values of the design parameters. It is to be noted that design parameter space is a generalized representation. The nature of this space varies depending on the design phase. For instance, during the preliminary design phase, states may represent feasible parameter values, while in the detailed design phase, they could be raw pixel images of engineering drawings or B-spline parameters in a Computer-Aided Design drawing file. This flexibility in state representation enables precise modelling of the design process within the MDP framework.

Action Space (A): A consists of all possible actions. In the engineering design context, an action refers to any decision made by the designer that changes the design state. Design decisions may involve updating values for one or more design parameters, and modifying materials or structural features. For example, a designer may increase the thickness of a component to enhance its strength or decrease a parameter to reduce weight. The action

space consists of all feasible actions available from a given design state and typically varies depending on the specific state.

Reward Function (R): The reward function provides a numerical score as feedback for taking a specific action from a given design state. The reward R depends on both the state and the action. When a design action (or update) modifies the design parameters, R evaluates the quality of the updated design. It is based on design attributes that measure how well the design meets the target functional requirements. A higher quality design, one that better satisfies these requirements, results in a higher reward.

For example, in aircraft design, a design that meets payload and cruising speed requirements while staying within cost constraints and safety standards will receive a higher reward. Conversely, a design that fails to meet one or more of these criteria would earn a lower reward. The formulation of the reward function is flexible. For instance, if an aircraft design fails to meet airworthiness standards, the reward could be set to zero, regardless of its performance in payload and cost requirements.

Mathematically, $R : S \times A \rightarrow \mathbb{R}$. Domain of reward function is the set of all possible actions for any state in the state space of the MDP. R maps the domain to a real number, with a higher value denoting a higher reward.

Transition Probabilities (T): T captures the probabilities of moving from one state to another as a result of taking a specific action. These probabilities account for inherent uncertainties in the design process, such as manufacturing variability, material properties, and external factors. By modelling these uncertainties,T helps predict the potential outcomes of different design decisions, facilitating more informed decision-making.

Discount Factor (γ): γ determines the relative importance of future rewards compared to immediate rewards. In the context of design, this factor influences the long-term strategic planning of the design process. A higher discount factor places greater emphasis on future rewards, encouraging designers to consider the long-term implications of their decisions. In contrast, a lower discount factor prioritizes immediate gains.

Observation Space (O): O includes the set of conditional observation probabilities, modelling the likelihood of different observations given the true design state. Engineering

design is inherently a collaborative process involving multiple disciplines and teams, each controlling only a subset of the overall design parameters and often lacking full visibility into the entire design state. By incorporating partial observability, the MDP framework accurately reflects the distributed nature of the design process and the limited information available to individual designers.

2.4.2 Aircraft Design as MDP

Figure 1.1 presents the aircraft preliminary design process as an example to illustrate the engineering design process framed as a MDP.

The preliminary design process begins with selecting a baseline design, which is typically an existing design in the database that closely aligns with the functional requirements of the target application. While the baseline design may meet or closely approach some requirements, it often falls short in others. From this starting point, the design team iteratively performs two key actions: (a) updates one or more design parameters, and (b) evaluates the performance of the updated design against the target requirements. This cycle continues until the design fully satisfies all the requirements.

The design state at time t, denoted as S^t, describes values of preliminary design parameters at time t. Some examples of preliminary design parameters are fuselage length, fuselage form factor, propeller diameter, wing aspect ratio, etc. At time step t, the design state S^t is evaluated against design requirements such as speed, payload capacity, fuel efficiency, acoustic properties, airworthiness, and structural integrity. Based on this evaluation, a reward R_t is calculated for the design state using an analytical or empirical formula or by running a numerical model such as finite element analysis or computational fluid dynamics. A design that better meets the design requirements receives a higher reward value. The reward depends on the physics of the problem and the target requirements of the design, such as mission requirements, safety standards, budget constraints, etc. The designer observes the reward R_t and takes a design action A^t. A design action updates the design. For example, increasing or decreasing one or more design parameters. Examples of such modifications include adjusting the fuselage length, altering the wing shape, or

changing the form factor to enhance aerodynamic efficiency. After the execution of the action A^t, the design transitions from the current state S^t to a new state S^{t+1}. The updated design is once again evaluated against the design requirements to calculate a new reward R^{t+1}.

The design state at time t, denoted as S^t, represents the values of the preliminary design parameters at that time point. Examples of preliminary design parameters include fuselage length, fuselage form factor, propeller diameter, wing aspect ratio, etc. At each time step $[t, t+1]$, the design state S^t is assessed based on design requirements, such as speed, payload capacity, fuel efficiency, acoustic properties, airworthiness, and structural integrity. A reward R_t is then calculated for the design state using an analytical or empirical formula, or by running a numerical model like finite element analysis or computational fluid dynamics. Designs that better meet the requirements are awarded higher reward values. The reward depends on the physics of the problem, aircraft mission requirements, safety standards, and budget constraints. After observing the reward R_t, the designer takes a design action A^t, which updates the design. This could involve modifying one or more parameters, such as adjusting fuselage length, changing wing shape, or altering the form factor to improve aerodynamic efficiency. After executing action A^t, the design transitions from the current state S^t to a new state S^{t+1}, which is then evaluated again to calculate a new reward R^{t+1}. The process continues until the design team produces a design which satisfactorily meets all the target requirements.

This iterative process of evaluating the current state, taking actions, transitioning to new states, and receiving rewards enables continuous refinement and optimization of the aircraft design, starting from a baseline. The entire process is an MDP, beginning at time $t = 0$ with the initial design state S^0, which corresponds to the baseline design. The process continues until the design achieves a state that meets the target requirements, indicated by a sufficient reward.

2.5 Solving MDP

Solving an MDP involves finding a mapping function from the observation space (O) to the action space (A) that maximizes the expected cumulative reward. The mapping function is called a policy. A policy is an optimal strategy that dictates the best actions to take in each step to maximize the cumulative reward. Mathematically, it is expressed as

$$\Pi : O \rightarrow A \tag{2.1}$$

Solving MDP is about determining the optimal policy that guides decision-making to achieve the best possible outcomes based on the defined rewards. In engineering design, this means systematically exploring design modifications to arrive at a solution that best satisfies all performance and requirement criteria.

In a fully observable setting, state values are fully observed. Therefore, a policy is a mapping function from state space (S) to the action space (A). Mathematically,

$$\Pi : S \rightarrow A \tag{2.2}$$

Two common approaches for solving MDPs are reinforcement learning and imitation learning. In imitation learning, the agent learn from expert trajectories or labeled datasets, mapping inputs to known outputs, which is particularly valuable in design optimization tasks with abundant data. In contrast, reinforcement learning involves neural networks interacting with an environment to learn optimal strategies through trial and error, which is essential when data is scarce or incomplete. Each has its advantages and challenges, and the choice between them often depends on the specific task and available resources. These methodologies—both imitation and reinforcement learning Methods— offer distinct advantages based on the nature of the data and the specific requirements of the engineering design task at hand.

2.5.1 Reinforcement Learning

Reinforcement learning is a type of method where systems learn on their own by trying different actions, learning from the outcomes, and improving over time through trial and

error. This is a trial-and-error approach where the agent interacts with the environment, and receives a reward as a feedback signal based on its actions. The agent learns the best actions through repeated interactions and feedback signals.

Advantages of reinforcement learning solution to MDP include:

- Self-learning: The agent learns through autonomous interactions with the environment using reward signals. There is no need for labelled data, which can be expensive and time-consuming to collect.
- Exploration: The agent can learn optimal policies even for unexplored or uncertain settings.

Despite these advantages, reinforcement learning is impractical in engineering design applications. Firstly, reinforcement learning is sample inefficient. Reinforcement learning often requires a large number of interactions with the environment to converge on a good policy. This is impractical in real-world engineering design applications as individual interactions are expensive and time-consuming. Each interaction requires computing a reward value which may involve running expensive computational models. A single stress analysis can sometimes take hours to run. Moreover, reinforcement learning algorithms are notorious for being unstable. The policy training process often gets stuck in suboptimal solutions or fails to converge.

2.5.2 Imitation Learning

Imitation Learning involves learning a policy by observing and mimicking expert demonstrations of a task, instead of relying on trial-and-error approaches. It allows an agent to learn a task by observing an expert or a set of demonstrations and mimicking the expert's behavior[Hussein et al. (2017)]. The agent learns to copy the expert's actions by studying a dataset of state-action pairs, allowing it to bypass the need for independent exploration of the environment.

Imitation learning algorithms can be highly data-efficient since the agent does not need to explore the environment on its own. It directly learns from expert examples. The

training process is more stable as they directly learn from expert trajectories. This technique is particularly valuable for engineering automation via solving MDP due to its ability to leverage existing human expertise and improve design optimization.

The main challenge associated with imitation learning is obtaining expert data. The success of imitation learning depends on obtaining high-quality expert trajectories that is relevant to the problem. We address that challenge in this thesis.

The next two sections describes imitation learning and reinforcement learning in further detail.

2.6 Imitation Learning

2.6.1 Types of Imitation Learning

Imitation learning encompasses several key approaches, each with unique strengths and potential applications within engineering design automation context. The popular types of imitation learning are behavioral cloning, inverse reinforcement learning, and generative adversarial imitation learning.

- **Behavioral Cloning:** Behavioral Cloning represents the simplest form of imitation learning, where an agent directly mimics the actions of an observed expert. This approach is particularly efficient for tasks with readily available demonstrations, allowing the agent to replicate successful design choices effectively. The agent learns from past successful designs and applies these strategies in similar situations. However, this method has its limitations, particularly in its ability to adapt to unforeseen or novel scenarios. If an unexpected component size or functionality requirement arises, the agent trained solely through behavioral cloning may struggle to make the necessary adjustments [Torabi et al. (2018), Kelly et al. (2019)]. As highlighted by [Osa et al. (2018)], while behavioral cloning is a powerful tool for direct imitation, it lacks the flexibility required for dynamic environments often encountered in engineering design.

- **Inverse Reinforcement Learning (IRL):** offers a more nuanced approach by inferring the reward function that motivates an expert's behavior based on their demonstrations. This method allows the agent to discern the underlying objectives guiding expert decisions, enabling more adaptive behavior in novel situations. In the context of design automation, IRL could be particularly valuable for analyzing successful designs to uncover the reward functions that human designers prioritize—such as minimizing weight or maximizing heat dissipation. By utilizing this inferred reward function, the agent can explore the design space more effectively and identify optimal solutions that align with the designer's goals [Ng et al. (2000), Abbeel and Ng (2004)]. This ability to adapt based on inferred objectives positions IRL as a powerful alternative to traditional imitation learning methods.

- **Generative Adversarial Imitation Learning (GAIL)** is a type of imitation learning that employs an adversarial framework to enhance the imitation process. GAIL utilizes a two-model system. First, a generative model (actor) that attempts to replicate the expert's behavior. Second, a discriminative model (critic) that distinguishes between the actions of the agent and real demonstrations. This adversarial setup creates a competitive environment where the agent is continually challenged to improve its imitation skills. Within engineering design automation, GAIL is particularly advantageous for intricate design tasks, where subtle nuances in design choices can significantly impact performance. The generative model, which is consistently pushed by the critic to refine its imitation, learns to replicate these finer details, leading to more accurate and optimal design solutions [Ho and Ermon (2016a)]. As emphasized by [Osa et al. (2018)], GAIL effectively combines the benefits of imitation learning with adversarial training, solidifying its position as a valuable approach for learning from expert behavior in engineering design contexts.

2.6.2 Pros and Cons of Imitation Learning Methods for Design Automation

Imitation learning approaches offer several benefits, including the ability to integrate human expertise directly into automation, improve optimization, and enhance the adaptability of design automation systems. Table 2.1 highlights key features of three imitation learning methods - behavioral cloning (BC), IRL, and GAIL - described in Section 2.6.1. Each method brings distinct advantages and challenges, making it essential to select the most appropriate IL approach based on the specific requirements of the design task.

Table 2.1: Types of Imitation Learning

	Types of Imitation Learning		
Features	BC	IRL	GAIL
Complexity	Low	High	High
Computational Cost	Low	High	High
Interpretability	High	Moderate	Low
Guaranteed Feasibility	High	Moderate	Moderate

Complexity: Imitation learning methods vary in their complexity, with BC being the simplest to implement, as it directly replicates expert actions. In contrast, IRL and GAIL involve more intricate processes, such as inferring reward functions or using adversarial networks, making them more computationally demanding [Osa et al. (2018)].

Computational Cost: BC is often more computationally efficient compared to methods like GAIL, which require iterative training between the actor, and critic models. As noted by [Ho and Ermon (2016b)], GAIL's adversarial setup tends to increase computational overhead, though it can produce more refined and adaptable models in return.

Interpretability: IRL offers greater interpretability compared to BC or GAIL, as it attempts to uncover the underlying reward functions driving expert behavior. This allows for better understanding and analysis of the design strategies used by human experts, which is particularly valuable in high-stakes design tasks where interpretability is crucial [Ng et al. (2000)].

Feasibility: All three methods have varying degrees of feasibility depending on the design

task. BC is highly feasible for straightforward tasks where expert demonstrations are readily available, but it may struggle with unseen situations. IRL and GAIL, while more complex, offer greater adaptability in dynamic environments where the agent needs to generalize beyond the specific demonstrations it has observed [Bouteiller (2022), Goo and Niekum (2023)].

After evaluating the features of these IL approaches, we decided to use BC due to its simplicity, computational efficiency, and direct applicability make it suited for automating design processes in environments where the expert's actions can be closely followed. However, for tasks requiring more flexibility and adaptability, advanced methods like IRL and GAIL offer superior performance, especially in scenarios with greater uncertainty [Bouteiller (2022), Goo and Niekum (2023), Bougie et al. (2023)]. While advanced techniques such as GAIL and IRL offer significant adaptability in dynamic environments, the straightforward nature and efficiency of BC make it particularly effective for design tasks. In the following section, the focus is placed on BC as the primary method employed in this thesis for automating the engineering design process.

2.6.3 Behavioral Cloning

BC enables an agent to learn by mapping observed states to corresponding actions. The agent observes the decisions made by the expert in various scenarios—whether navigating a maze or making strategic choices—and learns to associate each state (such as turns, decisions, or specific movements) with the correct response. This approach builds on supervised learning, where the goal is to minimize the error between the agent's predicted actions and those performed by the expert. A loss function is employed to measure the accuracy of the agent's learning, calculating the difference between the predicted actions from a policy and the actual actions demonstrated. As training progresses, the agent gradually refines its policy, minimizing the loss function and more closely replicating the expert's actions [Ho and Ermon (2016a), Sutton and Barto (2018)].

Although BC's simplicity makes it an efficient method for training agents, careful attention must be given to the type of data used during training. When working with sequential

data, where each action depends on the previous one—such as a series of decisions in a complex design process—the use of specialized neural network architectures becomes crucial. Recurrent neural networks, for example, are particularly well-suited for handling such dependencies, as they retain information from previous actions and use it to inform current decisions. This ability to capture temporal patterns is vital in engineering design automation, where past design choices often influence subsequent decisions [Hochreiter and Schmidhuber (1997)].

Having examined the advantages and limitations of BC in the context of engineering design automation, it is essential to delve into the various architectures employed in BC implementations. The choice of architecture plays a pivotal role in determining the effectiveness of BC, as different neural network structures can significantly impact the agent's ability to learn from expert demonstrations. Understanding these architectural considerations not only highlights the practical aspects of applying BC but also addresses how specific designs can enhance the model's performance and adaptability in complex engineering tasks. In the following section, the key BC architectures will be explored, focusing on their characteristics, advantages, and suitability for various design applications.

2.6.4 Architectures of Behavioral Cloning

BC leverages various neural network architectures to replicate the decision-making process of expert agents. The choice of architecture is critical as it determines how well the model can learn and generalize from the provided demonstrations. For simple, non-sequential tasks, feedforward neural networks are often sufficient. However, for more complex scenarios involving temporal dependencies and sequential data, advanced architectures like simple-Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and Gated Recurrent Units (GRUs) are preferred. These architectures are designed to capture and process information across different time steps, enabling the agent to make informed decisions based on past experiences. This section explores the different neural network architectures employed in BC, highlighting their strengths and suitability for various types of tasks [Pomerleau (1988), Hochreiter and Schmidhuber (1997), Cho et al. (2014a), Goodfellow

et al. (2016)].

Long Short Term Memory: Long Short-Term Memory (LSTM) networks are a specialized type of recurrent neural network (RNN) designed to learn and remember long-term dependencies in sequential data. In the context of imitation learning, LSTMs can be viewed as a method to mimic complex temporal patterns exhibited by expert demonstrators. LSTMs excel at capturing the nuanced, time-dependent decision-making processes of human experts. By observing sequences of expert actions and corresponding states, LSTM-based models can learn to replicate intricate behaviors over extended time horizons, making them particularly suitable for tasks requiring long-term memory and context understanding.

The key innovation of LSTMs lies in their unique cell structure, which includes:
Input Gate: Controls what new information is added to the cell state.
Forget Gate: Decides what information should be discarded from the cell state.
Output Gate: Determines what information from the cell state should be used as output.
Cell State: A memory component that runs through the entire chain of LSTM units.
This gated architecture allows LSTMs to selectively remember or forget information over long sequences, enabling them to maintain relevant context and discard irrelevant details. Also you can find the shape of LSTM cell below[Sahraei et al. (2021)]:

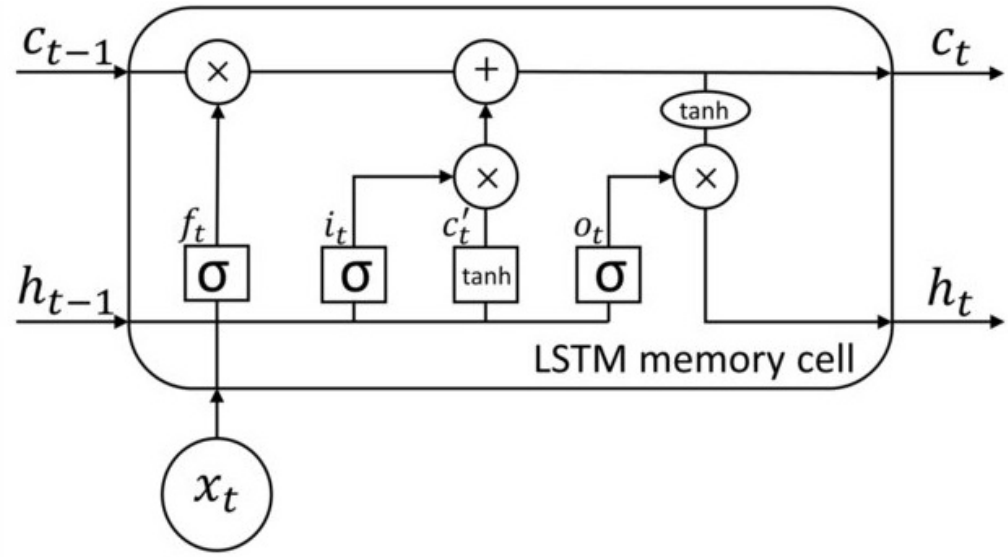


Figure 2.2: LSTM Cell Structure

Long Short-Term Memory (LSTM) cells are a type of recurrent neural network unit designed to capture long-range dependencies in sequence data by maintaining an internal cell state

and controlling information flow through input, forget, and output gates. The input gate (i_t)regulates the amount of new information added to the cell state, while the forget gate (f_t)determines how much of the previous state (c_{t-1}) is retained. The output gate (o_t) decides how much of the cell state is used to compute the output (h_t). These gates utilize sigmoid activations to produce outputs between 0 and 1, effectively gating information flow, while the cell state updates use the hyperbolic tangent function to ensure that the state values remain within a suitable range. This structure allows LSTMs to effectively manage vanishing and exploding gradient issues, making them particularly useful for tasks involving long-term dependencies [Hochreiter and Schmidhuber (1997)].

Pros and Cons of Using LSTM:

The advantages and disadvantages of LSTM networks are detailed below.

Pros:

1. **Long-term Dependency Handling:** LSTMs excel at capturing long-range dependencies in sequential data.[Hochreiter and Schmidhuber (1997)].
2. **Vanishing Gradient Mitigation:** The gated structure helps alleviate the vanishing gradient problem common in traditional RNNs [Gers et al. (2000)].
3. **Flexibility:** LSTMs can be applied to a wide range of sequence modeling tasks [Graves (2014)].
4. **Context Awareness:** They can maintain relevant context over long sequences [Greff et al. (2017)].

Cons:

1. **Computational Complexity:** LSTMs can be computationally intensive, especially for very long sequences [Hochreiter and Schmidhuber (1997)].
2. **Training Difficulty:** They may require large amounts of data and careful hyperparameter tuning [Goodfellow et al. (2016)].
3. **Limited Parallelization:** The sequential nature of LSTMs can limit parallelization opportunities [Joulin and Mikolov (2015)].

4. **Fixed Memory Size:** The cell state has a fixed size, which may be a limitation for some applications [Britz et al. (2017)].

In the context of imitation learning, LSTMs provide a powerful tool for capturing and replicating complex temporal behaviors, making them valuable in applications ranging from robotic control to natural language processing [Van Houdt et al. (2020), Sun et al. (2017)].

Gated Recurrent Unit: Gated Recurrent Units (GRUs) are a type of recurrent neural network architecture designed to capture long-term dependencies in sequential data. GRUs can effectively model and replicate complex temporal behaviors exhibited by expert demonstrators in imitation learning tasks.

Structure of GRU Cells

The GRU cell structure consists of two main components:

Reset Gate: Determines how much of the past information to forget. Update Gate: Decides what information to throw away and what new information to add. Unlike LSTMs, GRUs don't have a separate memory cell, instead directly updating their hidden state.

Also, the cell shape of the GRU is below[Li et al. (2021)]:

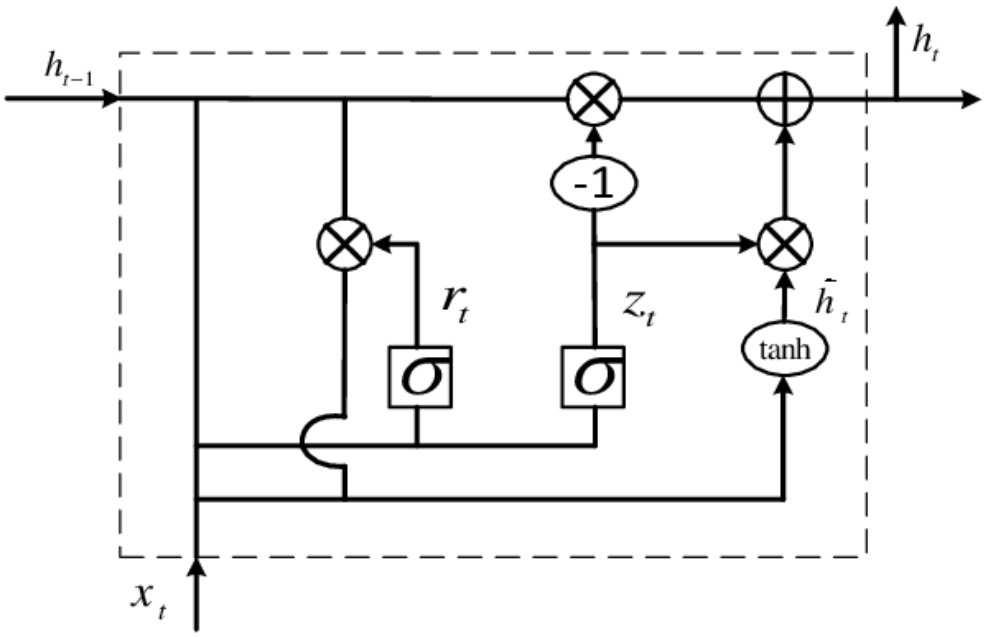


Figure 2.3: GRU Cell Structure

The diagram represents the structure of a Gated Recurrent Unit (GRU) cell, a type of re-

current neural network designed to address the vanishing gradient problem associated with traditional RNNs. The GRU cell utilizes two gating mechanisms: the reset gate (r_t) and the update gate (z_t). The reset gate controls the influence of the previous hidden state (h_{t-1}) on the candidate hidden state (h_t), allowing the model to forget irrelevant past information. In contrast, the update gate determines the extent to which the candidate hidden state (h_t) contributes to the current hidden state (h_t), balancing between retaining past knowledge and incorporating new information. These gates are computed using the sigmoid activation function, which outputs values between 0 and 1, while the candidate hidden state employs the hyperbolic tangent function to allow for a range of values between -1 and 1. By dynamically adjusting the flow of information through these gates, the GRU effectively captures temporal dependencies in sequential data, providing a simpler and computationally efficient alternative to other recurrent architectures.

Pros and Cons of Using GRU

The advantages and disadvantages of GRU networks are detailed below.

Pros:

1. **Simpler Architecture:** GRUs have fewer parameters than LSTMs, making them computationally more efficient [Cho et al. (2014b)].

2. **Easier to Train:** The simpler structure often leads to faster training times [Chung et al. (2014)].

3. **Good Performance:** Despite their simplicity, GRUs often perform comparably to LSTMs on many tasks [Chung et al. (2014)].

Cons:

1. **Less Powerful for Very Long Sequences:** In some cases, GRUs may not perform as well as LSTMs for very long sequences [Chung et al. (2014)].

2. **Less Flexibility:** The simpler structure might limit GRUs' ability to model certain complex patterns that LSTMs can capture [Chung et al. (2014)].

In the context of imitation learning, GRUs provide a computationally efficient alternative to LSTMs for capturing and replicating temporal behaviors, particularly useful in applications where sequence lengths are moderate and computational efficiency is a priority.

Simple Recurrent Neural Network: Simple Recurrent Neural Networks are a class of artificial neural networks designed to process sequential data. In the context of imitation learning, simple RNNs can be used to model and replicate temporal patterns in expert demonstrations, though they are less sophisticated than their successors like LSTMs and GRUs.

Structure of Simple RNN Cells

The structure of a simple RNN cell consists of:

1. Input Layer: Receives the current input in the sequence.
2. Hidden Layer: Maintains the network's memory state.
3. Output Layer: Produces the network's output for the current time step.

The key feature of simple RNNs is the recurrent connection, where the hidden state at each time step is influenced by both the current input and the previous hidden state.
Also, the cell shape of Simple RNN is like below:

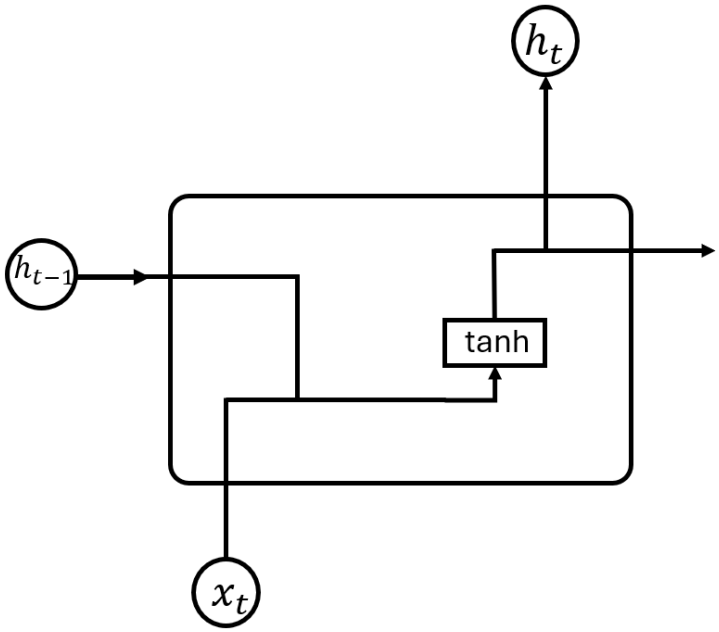


Figure 2.4: Simple RNN Cell Structure

Pros and Cons of Using Simple RNN

The advantages and disadvantages of Simple RNN networks are detailed below.

Pros:

1. **Simplicity:** Easy to understand and implement.
2. **Efficiency:** Computationally less expensive than more complex architectures [Greff et al. (2017)].
3. **Suitable for Short Sequences:** Effective for tasks involving short-term dependencies [Lipton et al. (2015)].

Cons:

1. **Vanishing/Exploding Gradients:** Struggle with learning long-term dependencies due to these issues [Hochreiter and Schmidhuber (1997)].
2. **Limited Memory:** Cannot effectively retain information over long sequences [Greff et al. (2017)].
3. **Less Powerful:** Generally outperformed by more advanced architectures like LSTMs and GRUs in complex tasks [Hochreiter and Schmidhuber (1997)].

Figure 2.4 illustrates the architecture of a simple Recurrent Neural Network. In this setup, the network tackles inputs one time step at a time, allowing it to remember information from previous steps through a recurrent connection. At each time step t, the network takes in an input x_t, which it combines with the previous hidden state h_{t-1} to update the current hidden state h_t. This recurrent connection gives the RNN a sort of memory, enabling it to learn temporal patterns by passing information from earlier steps to inform future computations.

The activation function employed here is the hyperbolic tangent. The updated hidden state h_t not only captures new information from the current input but also retains important details from previous time steps, effectively encoding the sequence history. This hidden state can then serve as the output for the current time step or be passed on to subsequent layers for further processing.

However, simple RNNs encounter challenges when it comes to long-term dependencies due to the vanishing gradient problem. This issue arises when the influence of earlier time steps diminishes as the network processes the sequence, making it difficult for the RNN to learn patterns that span longer sequences. This limitation can hinder the network's effectiveness in tasks that require retaining long-term context. In imitation learning contexts, simple RNNs can be useful for tasks involving short-term patterns or when computational resources are limited.

The computational complexity of recurrent architectures plays a crucial role in their practical applicability to engineering design automation. While Simple RNNs offer lower computational costs, they struggle with long-term dependencies due to vanishing gradients. LSTMs and GRUs address this limitation through gating mechanisms, but at the expense of increased computational overhead. LSTMs, with their four gating mechanisms, are the most computationally expensive, whereas GRUs strike a balance between efficiency and performance by using fewer gates. These trade-offs highlight the need to carefully select an architecture that balances accuracy, efficiency, and available computational resources when applying sequence-learning models to engineering design tasks.

Complexity Analysis

In this section, we analyze the complexity of three prominent recurrent neural network (RNN) architectures: Simple RNNs, Long Short-Term Memory networks (LSTMs), and Gated Recurrent Units (GRUs). Understanding their time and space complexities is crucial for selecting the appropriate model based on the computational resources available and the specific requirements of a task.[Cahuantzi et al. (2023)].

Time Complexity: The time complexity of RNN architectures is primarily determined by the number of operations required per time step during both the forward and backward passes. This complexity is influenced by factors such as the size of the input vector, the number of hidden units, and the specific operations performed within each unit.

- **Simple RNNs**: In a Simple RNN, each time step involves computing the hidden state by applying a weight matrix to the input and the previous hidden state, followed by an activation function. The time complexity per time step is $O(n \cdot h + h^2)$, where n is

the input size and h is the number of hidden units.

- **LSTMs**: LSTMs introduce additional components, including input, output, and forget gates, as well as a cell state. Each gate involves separate weight matrices and element-wise operations. Consequently, the time complexity per time step increases to $O(4(n \cdot h + h^2))$, accounting for the computations associated with the four gates and the cell state

- **GRUs**: GRUs simplify the LSTM architecture by combining certain gates, resulting in fewer parameters. Specifically, GRUs utilize reset and update gates. The time complexity per time step for GRUs is $O(3(n \cdot h + h^2))$, reflecting the computations for the three gates.

Space Complexity:

Space complexity pertains to the memory requirements for storing model parameters and intermediate activations during training and inference.

- **Simple RNNs**: The primary space requirements stem from the weight matrices and bias terms. The space complexity is $O(n \cdot h + h^2)$, corresponding to the input-to-hidden and hidden-to-hidden weight matrices, respectively.

- **LSTMs**: Due to the additional gates and cell states, LSTMs require more parameters. The space complexity is $O(4(n \cdot h + h^2))$, accounting for the weight matrices associated with each gate and the cell state.

- **GRUs**: GRUs, with their streamlined architecture, have a space complexity of $O(3(n \cdot h + h^2))$, reflecting the parameters for the reset and update gates.

Comparison Summary

Table 2.2 summarizes the computational complexities of Simple RNNs, LSTMs, and GRUs:

When selecting an RNN architecture, it is essential to balance computational efficiency with the model's ability to capture temporal dependencies. Simple RNNs offer lower computational costs but may struggle with long-term dependencies due to issues like vanishing

Architecture	Time Complexity per Time Step	Space Complexity
Simple RNN	$O(n \cdot h + h^2)$	$O(n \cdot h + h^2)$
LSTM	$O(4(n \cdot h + h^2))$	$O(4(n \cdot h + h^2))$
GRU	$O(3(n \cdot h + h^2))$	$O(3(n \cdot h + h^2))$

Table 2.2: Computational Complexity Comparison of RNN Architectures

gradients. LSTMs and GRUs mitigate these issues through gating mechanisms, with LSTMs providing a more comprehensive solution at the cost of increased complexity, while GRUs offer a middle ground with reduced complexity and competitive performance[Cahuantzi et al. (2023)]

While supervised learning excels in data-rich environments, many engineering design tasks involve uncertainty or data scarcity, where labeled data is not readily available. In these cases, reinforcement learning presents a robust alternative. Unlike supervised learning, reinforcement learning does not rely on labeled input-output pairs. Instead, it learns through interaction with the environment, receiving feedback in the form of rewards or penalties based on its actions. Reinforcement learning is particularly well-suited for sequential decision-making problems, where each decision affects future outcomes. By iteratively exploring different strategies and refining its approach based on feedback, an reinforcement learning agent can optimize design processes even in the absence of large amounts of data.

2.7 Reinforcement Learning

Reinforcement learning algorithms are classified based on whether the algorithm focuses on learning a policy directly or learning a value function to guide decision-making. The two types are defined as follows:

1. Policy-Based Methods: These methods directly learn the policy function, mapping states to actions. They offer advantages like sample efficiency and potentially faster convergence in specific scenarios. However, they might struggle with balancing exploration and exploitation[Nachum et al. (2017b)].

2. Value-Based Methods: These methods prioritize estimating the value of states or state-action pairs. This value estimation helps guide the agent towards better actions. However, they often require learning an additional function (the value function) alongside the policy, potentially increasing computational complexity[Sutton and Barto (2018)].

Value-based and policy-based methods are two fundamental approaches that differ in how they learn optimal behavior. Value-based methods focus on estimating the value function, which represents the expected cumulative reward for each state or state-action pair. The optimal policy is then derived implicitly by selecting actions that maximize the estimated value. Examples of value-based algorithms include Q-learning and SARSA. In contrast, policy-based methods directly learn a policy function that maps states to actions, without necessarily estimating value functions. These methods optimize the policy parameters to maximize expected rewards, often using policy gradient techniques. REINFORCE is a classic example of a policy-based algorithm.

Each approach has its strengths and limitations. Value-based methods can be more sample-efficient and often achieve better performance in discrete action spaces. They also provide a clear interpretation of the agent's decision-making process through the learned value function. However, they struggle with continuous or high-dimensional action spaces and may have convergence issues in some scenarios. Policy-based methods, on the other hand, can naturally handle continuous action spaces and have better convergence properties. They also allow for learning stochastic policies, which can be beneficial in certain environments. However, policy-based methods typically have higher variance in gradient estimates and may require more samples to learn effectively.

In recent years, hybrid approaches like actor-critic methods have gained popularity by combining elements of both value-based and policy-based learning. These methods aim to leverage the strengths of both approaches while mitigating their individual weaknesses. The choice between value-based, policy-based, or hybrid methods often depends on the specific characteristics of the problem at hand, such as the nature of the action space, the complexity of the environment, and the available computational resources[Nachum et al.

(2017a)].

2.7.1 Benefits in Engineering Design Automation

Using reinforcement learning for automating Engineering Design, has some benefits like below:

Adaptability and Learning from Experience: RL algorithms can learn from past experiences and continuously improve their decision-making strategies. This adaptability is crucial in dynamic engineering environments where design requirements and constraints may change over time[Silver and Veness (2010)].

Real-Time Decision Making: RL enables real-time decision-making frameworks, allowing for quick adjustments to design parameters in response to new information or changing constraints. This capability enhances the responsiveness and flexibility of automated design systems, making them more agile and efficient in handling iterative design processes[S Silver and Veness (2010)].

MDPs provide a mathematical framework to model design processes as sequences of states, actions, and rewards. States represent design configurations, actions correspond to decisions or modifications, and rewards quantify the performance or quality of the designs. The objective is to derive an optimal policy that maximizes cumulative rewards over time[Sutton and Barto (2018), Bertsekas (2012)].

Reinforcement learning algorithms are well-suited for addressing MDP challenges in engineering design due to several key advantages:

- Handling Complexity: RL, especially when integrated with deep learning, manages high-dimensional state and action spaces efficiently[Mnih et al. (2015)].
- Dealing with Uncertainty: RL accommodates stochastic environments, making it ideal for uncertain or probabilistic design scenarios[Silver and Veness (2010)].
- Sequential Decision Making: RL aligns naturally with the iterative and sequential nature of engineering design processes[Sutton and Barto (2018)].

- Adaptability: RL agents adapt to dynamic changes in design requirements or environments[Powell (2007)].

While reinforcement learning demonstrates significant potential for solving MDP problems, challenges remain, such as the need for defining effective reward functions, and ensuring interpretability of learned policies[Wang et al. (2024), Dulac-Arnold et al. (2019)]. RL provides a robust framework for solving MDP problems in engineering design, the choice of RL approach—whether policy-based or value-based—can significantly impact performance and applicability. In particular, policy-based methods offer distinct advantages over value-based methods in certain design optimization scenarios, which are particularly relevant in the context of engineering design automation

2.7.2 Advantage of Policy-based over Value-based in EDA:

This section examines the reasons why policy-based methods often hold advantages for design automation tasks.

1. Exploration and Continuous Improvement: Engineering design problems often require continuous exploration of the design space to identify optimal solutions. Policy-based methods directly learn a policy mapping states to actions, making them well-suited for this exploration. The policy can adaptively explore new design configurations based on the rewards it receives, fostering continuous improvement. In contrast, value-based methods, while identifying good design configurations, may require additional steps to translate that knowledge into specific design choices. This can be inefficient for design automation, where continuous exploration is essential [Cao and ZhiMin (2019)].

2. High-Dimensional Design Spaces: Engineering design problems often involve high-dimensional design spaces encompassing various parameters. Policy-based methods can be more efficient in these scenarios. They directly learn a mapping from a complex state representation to a desired action, potentially bypassing the need for explicitly calculating value estimates for every possible state. Value-based methods, in contrast, often struggle with scalability in high-dimensional spaces. Calculating value estimates for every possible state can become computationally expensive and impractical [Sutton and Barto (2018)].

3. Interpretability and Explainability: Understanding the rationale behind design choices is crucial in engineering design, particularly for safety-critical applications. Policy-based methods, particularly those based on actor-critic architectures, can offer better interpretability compared to value-based methods. Actor-critic architectures learn both a policy and a value function, allowing for some understanding of the policy's decision-making process [Lillicrap et al. (2019)]. While interpretability techniques exist for some value-based methods, they might not be as readily interpretable as the policies learned by policy-based methods. This lack of interpretability in value-based methods can pose challenges when justifying design choices in critical engineering domains.

We select two reinforcement learning algorithms to investigate their effectiveness in engineering design:

- Proximal Policy Optimization (PPO) - which is a policy-based method.
- Actor-Critic (A2C) - which is a hybrid approach that combines both policy-based and value-based methods.

Each of these approaches has several benefits and key concepts and application that are reviewed below :

PPO is a widely used on-policy policy gradient algorithm known for its stability and sample efficiency. It addresses the challenge of balancing exploration (trying new actions) and exploitation (utilizing known good actions) in RL[Schulman et al. (2017)].

Policy Gradient: This method updates the policy in the direction of higher rewards.

Clipping: PPO utilizes clipping to ensure the new policy remains close to the old policy, preventing drastic changes and promoting exploration within a reasonable range.

Importance Sampling: PPO incorporates importance sampling to adjust for the difference between the new and old policies, leading to more accurate policy updates.

A2C is an on-policy actor-critic method that combines policy learning with value estimation. It utilizes two separate neural networks: an actor network for policy learning and a critic network for value estimation.

Actor Network: This network directly learns the policy function, mapping states to actions.

Critic Network: This network estimates the value function, indicating the expected future reward for a given state or state-action pair.

Advantage Function: The advantage function measures how much better an action performs compared to the average action for a particular state, guiding policy updates.

Chapter 3

Data Collection from a Surrogate Design Task

This chapter details the methodology and framework for collecting human design decision data using a web-based design studio. First, aircraft design task was formulated as a Markov decision process. Section 3.1 describes this formulation. The experimental platform was developed to simulate a decentralized engineering design task, enabling multiple participants to collaboratively design a surrogate aircraft by controlling independent subsystems. Through structured sessions involving engineering students and professionals, data on design decisions and outcomes were systematically gathered to facilitate behavioral analysis and modeling. Section 3.2 describes the experiment construct and data collection process. Section 3.3 provides a summary of the collected data.

3.1 Surrogate Design Task: Aircraft Design

This section explains the formulations and implementation of an electric aircraft design task as a surrogate engineering design task.

3.1.1 State Space of Aircraft Design

A typical aircraft design progresses through three phases: conceptual design, preliminary design, and detailed design, each differing in the level of design definition and detail. Preliminary design phases is the stage where the main design parameters are defined for the chosen concept. The surrogate aircraft design task models the preliminary design phase.

The preliminary design task is modelled as centered around four subsystems, which corresponds to different aircraft components:

1. fuselage
2. payload
3. propulsion
4. airfoil

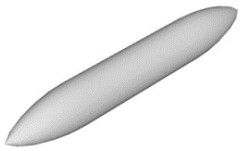
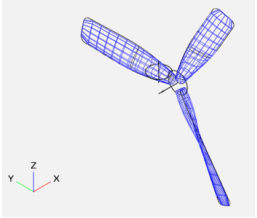
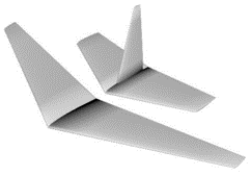
Subsystem	Design Parameters	Image	Symbol
Fuselage	Length Max Diameter Wing Location		x_1 x_2 x_3
Payload	Number Cells Charge per Cell Location		x_4 x_5 x_6
Propulsion	Diameter Rotation Rate Number Blades		x_7 x_8 x_9
Airfoil	Wingspan Root Chord Length Tail Scaling		x_{10} x_{11} x_{12}

Table 3.1: Subsystem Design Parameters, Images, and Symbols

A subsystem modelling captures a modular approach to a complex design process by breaking it down into manageable components, where each subsystem team (also called a design office) is responsible for one of the components. For example, the fuselage design office is only responsible for design parameters, such as fuselage length and max diameter. Table 3.1 summarizes the design parameters controlled by each of the four design offices. The parameters are described below:

- Fuselage design office manages the geometry of the tapered cylindrical vessel containing the payload through two parameters: length (x_1) and maximum diameter (x_2). The fuselage office is also responsible for the longitudinal location of the wing (x_3) expressed as a percentage of fuselage length measured from the aircraft nose.

- Payload design office is responsible for managing the battery system that powers the propulsion system by specifying the number of cells (x_4), charge per cell (x_5), and location (x_6). Location is the longitudinal position of the payload center of gravity measured from the aircraft nose expressed as a percentage of fuselage length.

- Propulsion design office oversees propeller selection and power requirements by specifying the propeller diameter (x_7), rotation rate (x_8, measured in rpm), and number of blades (x_9).

- Airfoil design office manages the geometry of the wing and tail surfaces by specifying the wing span (x_{10}), root chord length (x_{11}, and tail scaling factor (x_{12},

In summary, each design office controls three design parameters. Therefore, the surrogate design task consists of twelve design parameters (denoted as $x_1, x_2, \ldots, x_{12}$). The state space S is 12-dimensional, consisting of both continuous dimensions (for example, fuselage length) and discrete dimensions (for example, number of propeller blades).

The design parameters are bounded. The upper bound and lower bound of each of the 12 design parameters are summarized in Table 3.2. The bounds were chosen such that the design parameter space is broad enough to make the design task complex. Table 3.2 also shows the discretization for each design parameter within their bounds. For example, design parameter x_9 denotes the number of blades in the propeller. There are only four choices 1, 2, 3, and 4 blades. On the other hand design parameter x_{10}, which denotes the wingspan, can take any of the values 1.13, 1.14, . . . , 3.00. Any design parameter value within the bounds corresponds to a design state in the state space.

Design Parameters	Lower Bound	Upper Bound	Increment	Unit
x_1	0.1	3.00	0.01	meter
x_2	0.01	1.00	0.01	meter
x_3	27	75	1	percentage
x_4	1	12	1	-
x_5	0.00	6.00	0.1	A h
x_6	25	75	1	percentage
x_7	0.25	1.5	0.01	meter
x_8	1000	5000	50	rpm
x_9	1	4	1	-
x_{10}	0.05	3.00	0.01	meter
x_{11}	0.05	1.75	0.01	meter
x_{12}	0.1	1.00	0.001	-

Table 3.2: Bounds and discretization for the twelve design parameters.

3.1.2 Reward Formulation for Aircraft Design Task

In an MDP reward function is a mapping from each state to a numerical value that tells how much reward is gained from being in that state. In engineering design, we define reward to be a numerical value that quantifies the quality of that design. We formulate design quality quantitatively based on how well the design meets the target requirements, which are called functional requirements. There are six functional requirements: endurance, net lift, net thrust, net moment, net volume, and cost.

In an MDP, the reward function assigns a numerical value to each state, indicating the reward associated with being in that state. In engineering design, this reward represents a quantitative measure of design quality, based on how well the design meets target requirements—known as functional requirements. For example, in aircraft design task the functional requirements are aircraft mission requirements, such as endurance, cruising speed, etc.

In the surrogate design task, six main functional requirements define design quality

at the system level: endurance, net lift, net thrust, net moment, net volume, and cost. As described in Section 3.1.1, the design task is organized into four subsystems: fuselage, payload, propulsion, and airfoil. Each subsystem's design quality is evaluated according to its own functional requirements, while the six main functional requirements are evaluated at the system level. These system-level requirements depend on the parameters of each subsystem and their interdependencies. The formulation of subsystem-level and system-level requirements is detailed below[Thekinen and Grogan (2021)].

Fuselage Subsystem-level Functional Requirements

The functional requirements (FRs) for the fuselage subsystem are mass (y_1), drag (y_2), and volume (y_3).

Fuselage drag is calculated using empirical models. At a constant cruising speed, the zero-lift drag coefficient is given by [Sadraey et al. (2012)]:

$$C_{D_o}^f = K\left(1 + \frac{60}{(x_1 x_2)^3} + \frac{0.0025}{x_1 x_2}\right)\frac{x_1}{x_2}$$

where K is a constant estimated from CFD analysis of the fuselage geometry. From $C_{D_o}^f$, drag is given by

$$y_2 = \frac{1}{2}\rho_{air} v^2 S_{ref}^f C_{D_o}^f$$

where $\rho_{air} = 1.225$ kg/m^3 is the density of the air at the cruising altitude, v is the aircraft cruising speed, and S_{ref} is the reference area of the geometry which we estimate using panel methods in SUAVE[Lukaczyk et al. (2015)]. The fuselage mass and volume

$$y_1 = \pi x_1 x_2 \rho_{skin} \rho_{thk}$$
$$y_3 = \pi x_1 \left(\frac{x_2}{2}\right)^2$$

approximate the fuselage as a hollow cylinder with thickness $\rho_{thk} = 3.175$ mm and material density $\rho_{skin} = 1700$ kg/m^3.

Payload Subsystem-level Functional Requirements

The functional requirements (FRs) for the payload subsystem are available energy in Watt-hours (y_4), required volume(y_5), and mass(y_6).

Available energy is given by

$$y_4 = x_4 x_5 V_{cell}$$

where $V_{cell} = 3.7$ V is the battery cell voltage. Payload mass is given by

$$y_6 = x_4 m_{cell}$$

where $m_{cell} = 0.057$ kg is the battery cell mass. Payload volume y_5 is given by

$$y_5 = y_6 / \rho_{cell}$$

where $\rho_{cell} = 3.8$ kg/m^3 is the battery cell density.

Propulsion Subsystem-level Functional Requirements

The functional requirements (FRs) for the propulsion subsystem are required power (y_7), mass (y_8), and generated thrust (y_9).

Propeller geometry is imported into SUAVE, where computational fluid dynamics (CFD) based on blade element theory is used to calculate the thrust and torque (denoted by M^p) produced by the propeller at cruising speed [MacDonald et al. (2017)]. The CFD model uses an assumed viscosity of $1.5 \cdot 10^{-5}$ N-s/m^2. The required power is given by

$$y_7 = M^p x_8$$

where power depends on the propeller's rotation rate. The mass of the propeller is modelled to increase linearly with the number of blades and proportional to the square of the propeller's diameter.

$$y_8 = 0.25 x_7^2 x_9$$

Airfoil Subsystem-level Functional Requirements

The functional requirements (FRs) for the airfoil subsystem are lift (y_{10}), drag (y_{11}), and mass (y_{12}).

Wing geometry from OpenVSP is imported into SUAVE [Lukaczyk et al. (2015)] to compute lift ($C_L^w(i)$) and drag coefficients ($C_D^w(i)$) using CFD analysis based on the vortex lattice

method. Additionally, the platform or reference area ($S^w_{ref}(i)$) for each aerodynamic element (such as wings and tail) is estimated using wing panel methods. The lift and drag forces are then calculated as follows:

$$y_{10} = \sum_i \frac{1}{2}\rho_{air} v^2 S^w_{ref}(i) C^w_L(i)$$
$$y_{11} = \sum_i \frac{1}{2}\rho_{air} v^2 S^w_{ref}(i) C^w_D(i)$$

where the cruising speed is set to $v = 22.22$ m/s. The mass of the wing is estimated as:

$$y_{12} = \sum_i \frac{1}{2}\rho_{air}\rho_{skin}\rho_{thk} S^w_{surf}(i)$$

where $S^w_{surf}(i)$ is the airfoil surface area, derived from wing geometry in SUAVE, with $\rho_{skin} = 1700$ kg/m^3 is the mass density, and $\rho_{thk} = 3.175$ mm is skin thickness.

System-level Functional Requirements

The six system-level FRs are endurance (y_{13}, in minutes), net lift (y_{14}, in newtons), net thrust (y_{15}, in newtons), net moment (y_{16}, Newton-m), net volume (y_{17}, in cubic-meters), and cost (y_{18}, in USD).

Endurance is calculated as

$$y_{13} = \frac{y_4}{y_7}$$

assuming that propulsion consumes all of the energy provided by the payload. Net lift is computed

$$y_{14} = y_{10} - 9.81(y_1 + y_6 + y_8 + y_{12})$$

This represents the difference between the lift generated by all airfoil surfaces and the weight force, neglecting fuselage lift, which is typically less than 5% of the total lift for the considered geometry [Sadraey et al. (2012)].

Net thrust is given by

$$y_{15} = y_9 - (y_2 + y_{11})$$

which is the difference between the thrust generated by the propeller and the total drag, excluding the drag induced by interactions between the wing and fuselage. The net moment,

calculated about the aircraft nose for simplicity, is:

$$y_{16} = M^{w}_{pitch} - 9.81x_1\left(y_6\frac{x_6}{100} + y_{12}\frac{x_3}{100} + y_1\frac{1}{2}\right) + x_1\frac{x_3}{100}y_{10}(1 + 0.2x_{12})$$

where M^{w}_{pitch} is the pitching moment generated by the wings and tails, obtained from CFD analysis. In the moment calculation, it is assumed that a tail with a tail-scaling factor (x_{12}) of 1 generates 20% of the moment produced by the wings. The net volume is calculated as:

$$y_{17} = y_3 - y_5$$

which represents the difference between the available and required volume. Finally, the cost is estimated as:

$$y_{18} = \rho_{cost}\,(y_1 + y_6 + y_8 + y_{12})$$

assuming $\rho_{cost} = 1.0$ USD/kg.

The significance and contribution of the six FRs in aircraft design process is outlines below:

Error endurance: This parameter evaluates the aircraft's capability to sustain flight over long periods without the need for refueling or recharging. It is crucial for optimizing the efficiency of the propulsion system, enhancing the aircraft's operational range and mission capabilities.

Error lift: Measures the efficiency of the aircraft's wings and aerodynamic surfaces in generating the lift necessary for flight. By minimizing this error, the aircraft can achieve and maintain flight with reduced drag and energy consumption, improving overall aerodynamic performance.

Error thrust: This parameter assesses the propulsion system's effectiveness in generating the thrust required to overcome drag and propel the aircraft forward. Optimizing this error leads to improved acceleration, cruising speeds, and fuel economy, ensuring the propulsion system works in harmony with the aircraft's aerodynamics.

Error moment: Focuses on the aircraft's stability and control by evaluating the balance of pitching, rolling, and yawing moments. Ensuring minimal error in this parameter guarantees that the aircraft maintains stability throughout its flight, reducing the risk of instability

Table 3.3: Normalized error constants.

Functional Req.	$y_i^{(N)}$	$y_i^{(L)}$	$y_i^{(U)}$
Endurance (y_{13})	570.7 min	30 min	9999 min
Net Lift (y_{14})	55.62 N	0 N	10 N
Net Thrust (y_{15})	464.58 N	0 N	10 N
Net Moment (y_{16})	618.31 N-m	−5 N-m	5 N-m
Net Volume (y_{17})	6.4 m^3	0 m^3	9999 m^3
Cost (Mass) (y_{18})	38.93 USD	0 USD	30 USD

or excessive control adjustments.

Error volume: Addresses the efficient use of internal and external volume in the aircraft's design, ensuring optimal space allocation for payloads, fuel, and batteries. An optimized y_{17} results in improved aerodynamic profiles, reduced drag, and better fuel efficiency by minimizing wasted space.

Error cost: Reflects the economic feasibility of the design, including the costs associated with materials, manufacturing, and operation. By balancing performance with cost efficiency, this parameter ensures that the aircraft design remains both high-performing and economically viable.

Normalized Error Formulation

The six system-level functional requirements $y_{13}, y_{14}, \ldots, y_{18}$ are jointly evaluated using a single error metric, which we call a normalized error (denoted as E). Equation 3.1 computes the normalized error using the six functional requirement values y_i with feasible range given by lower bound $y_i^{(L)}$ and upper bound $y_i^{(U)}$, and normalization factor $y_i^{(N)}$.

$$E = \frac{1}{6}\sum_{i=13}^{18} \frac{\max\left(y_i, y_i^{(U)}\right) - y_i^{(U)} + \min\left(y_i, y_i^{(L)}\right) - y_i^{(L)}}{y_i^{(N)}} \tag{3.1}$$

The normalization factor corrects is applied to address the differing scales of each functional requirement (for example, net moment, y_{16}, has an order of magnitude higher variability than net lift, y_{14}). Table 3.3 lists the values for $y_i^{(N)}$, $y_i^{(L)}$, and $y_i^{(U)}$ for each FR.

The normalization equation ensures that E is a number between 0 and 1, with a lower value corresponding to a design with a higher quality (or that is closer to meeting all the functional requirements). A design that fully satisfies all functional requirements achieves a normalized error of zero. Therefore, the design objective is considered complete when a solution with $E = 0$ is attained.

3.1.3 Starting State: Baseline Design

The aircraft design process starts with a baseline design—a preliminary configuration that serves as a foundation for iterative refinements. This approach is common in the design of custom products, such as aircraft and ships, where a baseline from an existing design database is chosen to closely match the initial requirements of the target design. The baseline design typically meets some, but not all, of the system-level requirements, providing a practical starting point.

The design process then follows an iterative cycle involving subsystem- and system-level analyses. Separate design teams handle each subsystem—fuselage, payload, propulsion, and airfoil. Subsystem adjustments are made in an inner loop, where parameters like wingspan, payload distribution, and propeller features are modified based on technical evaluations. Each iteration aims to optimize one or more functional requirements while preserving or enhancing previously achieved targets.

The baseline design was modelled using OpenVSP. Table 3.4 lists the values of the 12 design parameters for the baseline design. The baseline design satisfied two out of the six requirements: endurance and volume check. The baseline design had a normalized error of 0.118(i.e., $E = 0.118$)

3.1.4 Q-score

Section 3.1.2 presents the formulations that connect the twelve design parameters to a normalized error metric, E. We now introduce a new metric called the Q-score, or quality score, represented by Q, as defined in Equation 3.2. Here, E_b is the normalized error for the

Table 3.4: Baseline aircraft design parameters.

Subsystem	Design Parameter	Value
Fuselage	Length (x_1)	1.57 m
Fuselage	Max Diameter (x_2)	0.86 m
Fuselage	Wing Location (x_3)	50 %
Payload	Number Cells (x_4)	8
Payload	Energy per Cell (x_5)	4.1 A-hr
Payload	Location (x_6)	58.72 %
Propulsion	Diameter (x_7)	0.63 m
Propulsion	Rotation Rate (x_8)	2900 rpm
Propulsion	Number Blades (x_9)	2
Airfoil	Wing Span (x_{10})	2.39 m
Airfoil	Root Chord Length (x_{11})	1.15 m
Airfoil	Tail Scaling (x_{12})	2.0

baseline design, and $\min(E, E_b)$ selects the lower of E or E_b. Thus, if the current design has a higher error than the baseline, the baseline error is used.

$$Q = 1 - \frac{\min(E, E_b)}{E_b} \tag{3.2}$$

The Q-score ranges from 0 to 1, where a higher score indicates better design quality. The baseline design has a Q-score of zero, while a design that fully meets all functional requirements achieves a Q-score of 1.

3.2 Data Collection: Human Subject Experiments

3.2.1 Web-based Studio for Data Collection

A design studio interface was created to facilitate the surrogate aircraft design task, offering a user-friendly setup suitable for engineering students (sophomore level and above). The task is divided into four subsystems: airfoil, fuselage, payload, and propulsion. The software architecture enables four human designers to control each subsystem simultaneously,

with individual logins assigned to each user by subsystem. After logging in, each designer accesses a dedicated "subsystem office" front-end window. Figure 3.1 illustrates the airfoil subsystem interface. Vertical sliders allow the user to adjust the airfoil parameters: wing span (x_{10}), chord ((x_{11})), and tail scaling factor ((x_{12})). The designer can modify these parameters by adjusting the sliders and then clicking the "Update" button in the top-right corner, which sends the selected parameter values to the server for technical analysis. Back-end technical services, hosted in the server, contain all of the engineering models needed for technical analyses. The server processes the inputs and returns outputs, including 3D visualizations, which appear on horizontal sliders. For the airfoil subsystem, outputs are lift (y_{10}), drag (y_{11})m, and wing mass (y_{12}).

Each designer independently adjusts their subsystem's design parameters during a 150-second iteration. The timer, displayed in the top-right corner, indicates the time remaining in the current cycle. The subsystem designers can also communicate with each other through the chat interface shown in Figure 3.1. When time expires, the subsytem-iteration concludes, and all four designers are directed to the "system" tab.

The system tab, shown in Figure 3.2, displays the current values of six system-level functional requirements: endurance (y_{13}), net lift (y_{14}), net thrust (y_{15}), net moment (y_{16}), net volume (y_{17}), and cost (y_{18}). These values are derived from the latest subsystem-level parameters. Designers review the system-level requirements and signal readiness for the next round using a "Ready" button in the top-right corner. Once all four designers indicate readiness, a new iteration begins. Iterations continue until all system requirements are satisfied or the allotted design process time expires.

Our interface simulates a typical aircraft design process, where subsystem iterations represent the inner loop and system iterations represent the outer loop. Each design office controls only its assigned parameters, reflecting the decentralized nature of complex design tasks. The design offices review functional requirements periodically, mirroring the process of design reviews in engineering projects.

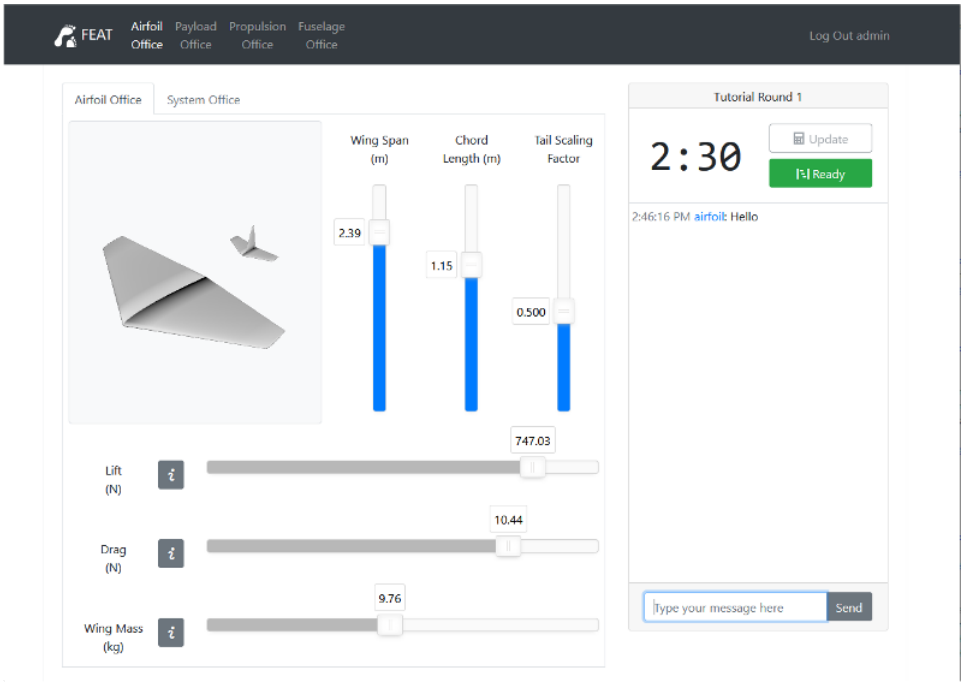


Figure 3.1: Subsystem tab of the web-based design interface.

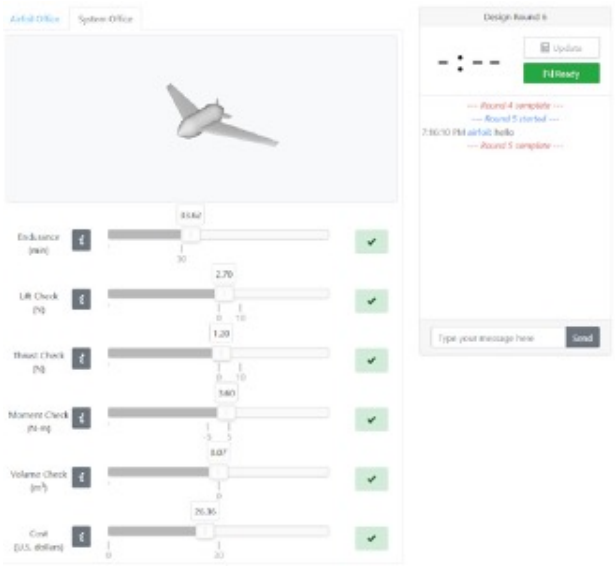


Figure 3.2: System tab of the web-based design interface.

3.2.2 Data Collection Sessions

The virtual design studio for the surrogate aircraft design task serves as a platform for collecting human design decision data for the task. This section outlines the procedure and data collection from human designers.

We conducted ten design sessions, each involving four participants (one per design

office). So, in total there were 40 participants. To be eligible, participants needed to be at least juniors in a STEM program and have prior engineering design experience. Most participants were graduate students with undergraduate engineering degrees. Each participant received a $10 gift card, plus a $5 bonus if the team could produce a design that met all six sytem-level functional requirements within the allotted session time.

Each session began with a baseline design that satisfied two of the six requirements: endurance and volume checck. The one-hour session was structured as follows:

- 10 minutes: Arrival and informed consent.
- 2 minutes: A pre-recorded orientation video explaining the virtual design studio, roles, and session rules.
- 8 minutes: Independent exploration of the interface.
- 10 minutes: Tutorial task simulating the design process, including timed design iterations and chat functions.
- 30 minutes: The main experimental aircraft design task.

Participants retained the same design role throughout the tutorial and main tasks to build familiarity with assigned design parameters and functional requirements. The tutorial and main design tasks used different cruising speeds to facilitate general problem understanding without transferring specific solutions. Each subsystem- and system-level phase had a maximum of 150 seconds per iteration, capping each complete iteration at 300 seconds (5 minutes). This ensured that at least six design iterations could occur within the 30-minute period. Participants completed subsystem- and system-level design iterations until they either met all six system-level functional requirements or reached the 30-minute time limit.

3.3 Data Collected

Table 3.5 presents a detailed comparison of the number of human design decisions and the Q-scores achieved for the ten human-subject design sessions. The Q-score serves as a

Table 3.5: Summary of number of human design decision data and normalized error from ten study sessions.

Session	Design Decisions	Q-score
1	123	0.449
2	172	0.797
3	30	0.949
4	121	1
5	154	0.669
6	105	1
7	177	0.992
8	183	0.288
9	155	0.669
10	182	0.322

quantitative measure of design quality, with a score of 1 indicating that the design meets all six system-level requirements. Out of the ten sessions, only two—Session 4 and Session 6—successfully achieved the design goal, attaining a Q-score of 1. Among these, Session 6 stood out as the highest-performing session. This team completed the design task with a relatively lower number of human design decisions, showcasing an efficient decision-making process while achieving optimal results. On the other hand, Session 8 was identified as the lowest-performing session, producing a design with the poorest quality and achieving a Q-score of just 0.288.

In total, the dataset comprises 1,402 design decision data points collected across ten sessions. Each data point captures the following information: the values of 12 design parameters ($x_1, x_2, ..., x_{12}$), the values of six functional requirements ($y_{13}, y_{14}, ..., y_{18}$), the system-level normalized error (E), and the associated Q-score.

The next chapter delves into the behavioral cloning study conducted on this dataset.

Chapter 4

Proposed Solution I: Imitation Learning

This chapter examines the use of behavioral cloning, a supervised imitation learning technique, to model and replicate human decision-making patterns. It investigates three sequence learning architectures to evaluate their performance. Finally, the chapter analyzes the effectiveness of a behavioral cloning-trained policy in automating engineering design tasks.

4.1 Accuracy in Modeling Human Design Decisions

We use a specific type of imitation learning called behavioral cloning, where an agent learns to replicate the actions of an expert by training on a dataset of observations (inputs) and corresponding actions (outputs) collected from the expert. It treats the learning process as a supervised learning problem. The architecture of behavioral cloning typically employs a neural network model designed to predict the action associated with a given state or observation.

In our approach, we utilized various sequence learning architectures for behavioral cloning to address the sequential and temporal nature of engineering design problems. Engineering design is inherently a sequential decision-making process, where current decisions depend on past states and actions. For instance, human designers adjust design parameters (e.g., propeller diameter) and observe their effects on functional requirements. After each iteration, feedback is received by running technical analysis models. These feed-

back loops, along with the memory of prior outcomes, inform subsequent design updates, making sequence learning architectures well-suited for capturing these dependencies.

We study and compare three sequence learning architectures: Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), and Simple Recurrent Neural Network (Simple-RNN). The architecture and results are described in this section.

4.1.1 LSTM

The input layer consists of 12 neurons, corresponding to the 12 design parameters used as input features. The LSTM layer, with 8 neurons, processes these inputs, capturing temporal dependencies in the data. We tested various numbers of neurons for the LSTM layer - such as, 4, 6, 8, 10, and 12. It was found that 8 neurons were the most suitable. The output layer containing 12 neurons predicts the set of 12 design parameters. This output layer is connected to the LSTM layer via a fully connected dense layer, enabling the model to map the learned features to the predicted outputs and the learning rate was used is 0.001 based on Adam optimize. The architecture comprises 780 trainable parameters in total. The activation function chosen for this neural network is ReLU, characterized by its zero output for negative inputs and a linear increase for positive values. The architecture is summarized in Table 4.1.

Table 4.1: LSTM Parameters.

Model : Sequential		
Layer(Type)	Output Shape	Params
LSTM	(None,8)	672
dense	(None,12)	108
Total Parameters	780	

In LSTM networks, lookback refers to the number of past timesteps the model considers as input to predict the future output. It defines the window size of sequential data that the LSTM processes at each step. In the design application of this thesis, a lookback of 5 means the model uses the design parameters from the previous 5 iterations to predict the

parameters for the next design step. The training and testing error corresponding to various lookback is summarized in Table 4.2. The errors are mean absolute error (MAE). The MAE is defined as the average absolute difference between predicted and actual values for each of the 12 design parameters, where the design parameters are normalized in the range 0-1. A lower MAE indicates higher accuracy in predicting the next value of the design decision, with values between 0 and 1. Based on the testing error, a lookback of 10 provided the most accurate predictions. This lookback indicates that the prediction of a design decision depends on the design decisions made during the previous 10 iterations. A properly chosen lookback ensures the model considers sufficient historical information without introducing unnecessary computational complexity.

Table 4.2: Training and test error of the LSTM based on number of lookback.

Lookback	Training Error	Testing Error
2	0.044	0.136
4	0.089	0.843
6	0.0652	0.634
8	0.0427	0.536
10	0.0424	0.1305
12	0.0681	0.631
14	0.0498	0.521
16	0.0722	0.253
18	0.0461	0.456
20	0.1035	0.632

A drawback of LSTM neural network is the high number of parameters required to achieve optimal results. The current network has approximately 780 parameters, as shown in Table 4.1. Given the excessive number of parameters, generalization with this method is very difficult. Therefore, an alternative neural network that retains memory capabilities but has fewer parameters was sought, leading to the GRU.

4.1.2 GRU

Another neural network was designed by replacing the LSTM layer with a GRU layer. This network retained the input layer with 12 neurons, representing the 12 design parameters, and the output layer, also with 12 neurons, to predict the same parameters for the next design step. The GRU layer had 8 neurons. Similar to the LSTM-based model, the output layer was connected to the GRU layer through a fully connected dense layer and the learning rate was used is 0.001 based on Adam optimize. Similar to LSTM, the ReLU activation function was chosen. The architecture is summarized in Table 4.3.

Table 4.3: GRU Parameters.

Model : Sequential		
Layer(Type)	Output Shape	Params
GRU	(None,8)	528
dense	(None,12)	108
Total Parameters	636	

Similar to LSTM, the number of lookbacks was tested from 2 to 20. The training and testing errors for various lookback are summarized in Table 4.4. Based on the testing error results, a lookback of 10 was found to be most accurate.

The advantage of GRU over LSTM lies in its fewer parameters, while still maintaining memory and flexibility. The fewer the overall number of parameters in a neural network, the easier it is to generalize the model.

4.1.3 Simple-RNN

A simple RNN-based neural network was also implemented with an input layer of 12 neurons for the design parameters and an output layer of 12 neurons for the predicted parameters. The RNN layer consisted of 8 neurons and processed sequences of input data. The output layer was connected to the RNN layer through a fully connected dense layer and the learning

Table 4.4: Training and test accuracy of the GRU based on number of lookback.

Lookback	Training Error	Testing Error
2	0.805	0.4161
4	0.087	0.6132
6	0.0464	0.5264
8	0.0426	0.6884
10	0.0413	0.1002
12	0.0804	0.4631
14	0.0422	0.6362
16	0.0425	0.3634
18	0.0512	0.4325
20	0.0439	0.5523

rate was used is 0.001 based on Adam optimizer. The number of parameters of the simple RNN is shown in Table 4.5.

Table 4.5: Simple-RNN Parameters.

Model : Sequential		
Layer(Type)	Output Shape	Params
Simple RNN	(None,8)	168
dense	(None,12)	108
Total Parameters	276	

To ensure consistent conditions with respect to lookback, the range of 2 to 20 was tested once again for Simple-RNN method. The results of these experiments are shown in Table 4.6. Once again, a lookback of 10 yielded the best accuracies as measured by testing error.

4.1.4 Comparing the Accuracy of Sequence Learning Architectures

Table 4.7 presents a comparative analysis of LSTM, GRU, and simple RNN models used for behavioral cloning, with the best mean absolute error as the metric. While all three archi-

Table 4.6: Training and test accuracy of the Simple RNN based on number of lookback.

Lookback	Training Error	Testing Error
2	0.0516	0.326
4	0.0511	0.147
6	0.0509	0.394
8	0.0506	0.281
10	0.0516	0.125
12	0.0506	0.943
14	0.051	0.269
16	0.0511	0.503
18	0.0501	0.643
20	0.0503	0.394

tectures leverage recurrent connections to process temporal information, the differences in their internal mechanisms, such as the gating structures in LSTM and GRU, influence their ability to capture long-term dependencies. The evaluation was conducted using a lookback window of 10, which was found to yield the most accurate predictions across all model types.

Table 4.7: Training and test accuracy of the three sequence learning methods for BC.

Approach	Training Error	Testing Error
LSTM	0.0424	0.13
GRU	0.042	0.10
Simple-RNN	0.052	0.12

The accuracy scores were evaluated for both the training and testing datasets. The LSTM and GRU models outperformed the simple RNN in terms of MAE on the training dataset. This can be attributed to their higher number of trainable parameters, which enhanced training accuracy. However, the LSTM model exhibited the lowest accuracy on the testing dataset, indicating weaker generalization. Notably, GRU achieved higher training accuracy than LSTM, despite having fewer parameters. This outcome may be due to the nature of

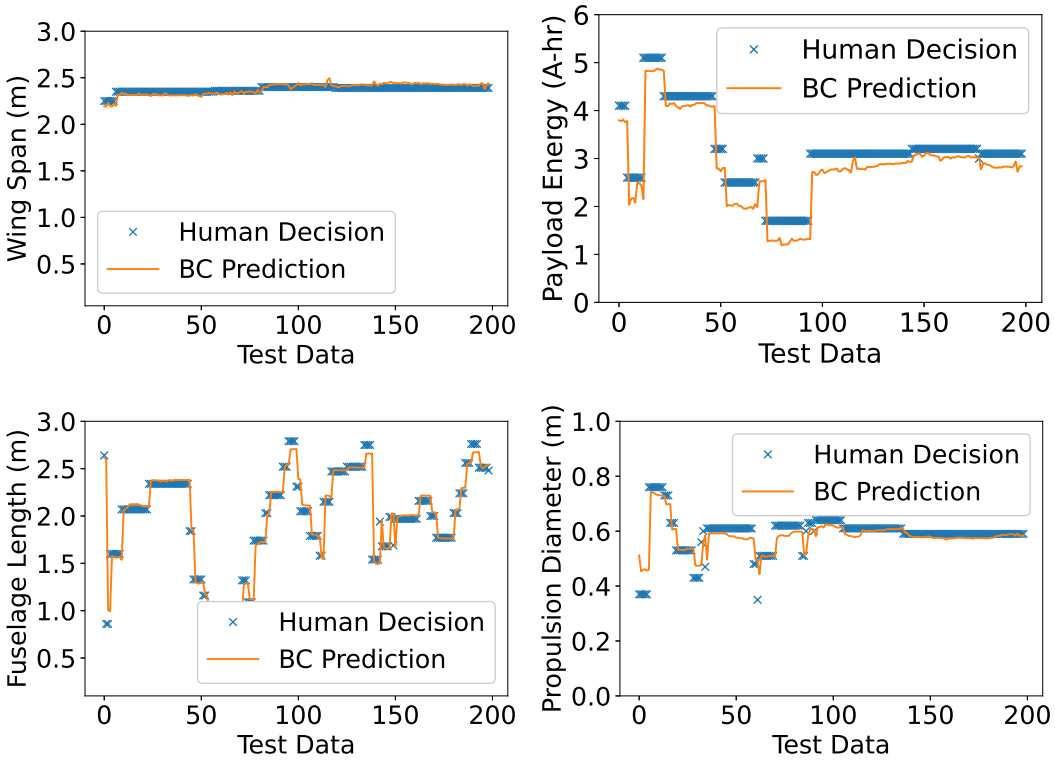


Figure 4.1: Comparing predictions from behavioral cloning policy (denoted by BC prediction) to decisions made by human designers.

the design task, where long-term dependencies were not critical. Among the three models, GRU demonstrated superior performance on both the training and testing datasets, making it the most effective choice for this application.

To further evaluate the performance of these architectures, a comparison was conducted between the predictions generated by the GRU model and the actual human decisions for four representative design variables. These variables were carefully selected to represent each of the four design offices involved in the study: wing span from the airfoil office, battery energy per cell from the payload office, fuselage length from the fuselage office, and propeller diameter from the propulsion office. This comparison was performed exclusively on the test dataset, which was intentionally excluded from the training process to ensure an unbiased evaluation of the model's generalization capabilities. The results of this comparison revealed that the GRU model accurately captured sequential human design decision-making patterns. In conclusion, the GRU architecture emerged as the most effective among the three models evaluated, consistently demonstrating superior performance on both the training and

testing datasets. This outcome underscores the potential of GRU-based behavioral cloning as a promising approach for modeling and automating human design decision-making processes in engineering contexts.

4.2 Evaluation of Agent Trained via Behavioral Cloning

This section presents a comprehensive analysis of imitation learning agent trained via behavioral cloning, with a focus on its ability to navigate the complex design space and iteratively improve design quality in the context of electric aircraft design. Specifically, we evaluated an imitation learning agent trained via the GRU architecture. GRU architecture was chosen as it gave the best performance in terms of testing error for predicting human design decisions.

To assess the agent's performance, 250 simulated design iterations were conducted. During these iterations, the imitation learning agent utilized its trained policy to make sequential design decisions, aiming to achieve higher-quality solutions. Each decision step involved selecting design parameters informed by prior iterations, with the agent leveraging its GRU-based model to capture and utilize temporal patterns in the design process.

The evaluation focused on the agent's ability to identify and transition toward improved design configurations while adhering to constraints imposed by the environment. The iterative approach allowed for the observation of the agent's learning dynamics and its capacity to generalize from the training data to new scenarios within the design space.

Figure 4.2 visualizes the Q-score for evaluating the quality of designs generated by the imitation learning agent. A Q-score of 0 represents the baseline design and signifies the starting point for the agent's exploration within the design space. As the imitation learning agent progresses through each iteration, it utilizes the policy derived from human design decision data to determine actions, thereby generating new design configurations. The ultimate goal of the RL agent is to produce a design that meets all the predefined design requirements, which is represented by a Q-score of 1. This score indicates that the agent has successfully identified an optimal design configuration that satisfies the constraints

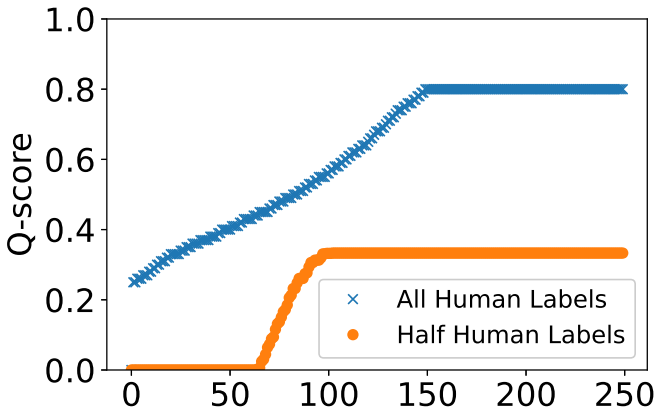


Figure 4.2: Q-score of forward simulation of 100 design iterations starting from baseline design. Baseline design has a Q-score of zero.

and objectives of the problem. The plot shown in Figure 4.2 captures the evolution of the Q-score across the 250 iterations. By tracking the trajectory of the Q-score, we can assess the agent's learning efficiency, its capacity to explore and exploit the design space, and its overall performance in achieving high-quality solutions.

The agent trained on human design decision data could generate a design with a Q-score of 0.8 after approximately 150 iterations. This indicates that the agent was able to explore the design space and improve the design quality significantly, approaching a solution that was close to the optimal design (with a score of 1). A score of 0.8 suggests a design that meets most, if not all, of the critical requirements, though there is still potential for further optimization. However, as the agent continued to interact with the environment, the Q-score plateaued, showing no further improvement in subsequent iterations. This stagnation in performance could indicate that the current policy was unable to explore the design space sufficiently to identify better solutions. This observation highlights a potential limitation in the learning process, where the agent, despite being trained on a comprehensive dataset, may struggle to fully explore and exploit the design space without further adjustments to the training process or the environment.

To investigate the impact of reducing the amount of human-labeled data, we trained a second RL agent using only half of the original human labels. By limiting the amount of available supervision, the agent was exposed to fewer design decisions from human experts, which could restrict its ability to learn an optimal policy. The results indicated that this

second agent achieved a maximum Q-score of 0.35 after approximately 100 iterations. This lower Q-score suggests that the agent, trained with less human guidance, was less effective at navigating the design space and generating high-quality solutions.

Chapter 5

Proposed Solution II: Reinforcement Learning

In the previous section, we observed that after approximately 150 iterations, the Q-score—the primary measure of design quality—reached a plateau, showing no further improvement. This stagnation highlights a critical limitation in the design process, suggesting that the dataset being utilized fails to support the discovery of improved design solutions beyond a certain point. This observation motivates the exploration of a hybrid approach that integrates RL and IL.

The hybrid RL-IL approach aims to overcome this plateau and enable the discovery of design solutions with higher Q-scores. RL offers the ability to explore and exploit the design space effectively, while IL leverages expert strategies to guide the learning process. Together, these methods hold the potential to achieve superior designs in a sample-efficient manner.

In this chapter, we investigate standalone RL and its integration with IL for the aircraft design MDP. Training an RL model for the aircraft design problem requires reward evaluations across thousands of states. As outlined in Chapter 3, reward evaluation for this problem is computationally intensive due to the need for CFD and FEA performed using SUAVE and OpenVSP modules. To address this challenge, we first trained a surrogate reward model using a neural network to approximate the reward (or design quality) based on the design parameters.

5.1 Surrogate Reward Model

There are six system-level FRs as described in Section 3.1.2: endurance (y_{13}, in minutes), net lift (y_{14}, in newtons), net thrust (y_{15}, in newtons), net moment (y_{16}, in N·m), net volume (y_{17}, in m^3), and cost (y_{18}, in USD). The formulations for these FRs are provided in Section 3.1.2.

The design quality is evaluated using a single metric, the normalized error E, as defined in Equation 3.1. E provides an overall assessment of the quality of the aircraft by integrating the performance of all subsystems, including propulsion, aerodynamics, fuselage, and payload subsystems.

The exact relationship between the design variables and these errors involves computationally expensive CFD and FEA models. Since neural networks are very good as general function approximators, we train a neural network based on the available data to capture the relation between the design variables and the six system-level FRs.

5.1.1 Training Data and Model Architecture

The dataset comprises 1567 samples, where each sample is represented as an (X, Y) tuple.

- X is a 12-dimensional vector, $X = [x_1, x_2, \ldots, x_{12}]$, representing the values of 12 design parameters. These parameters serve as the input features for the model.
- Y is a 7-dimensional vector, $Y = [y_{13}, y_{17}, \ldots, y_{18}, E]$. The first six elements of Y correspond to the six system-level functional requirements calculated based on the input representing the values of 12 design parameters. The seventh element represents the overall system error E associated with the design values X.

Each of the 12 design parameters in $X = [x_1, x_2, \ldots, x_{12}]$ was normalized independently using its respective maximum and minimum values in the dataset. This transformation centred the data around 0, ensuring that all feature values fall within the $[-1, 1]$ range. Each of the seven dimensions of $Y = [y_{13}, y_{17}, \ldots, y_{18}, E]$ was normalized using the same transformation. This ensured consistency in the scale of the functional requirements (FRs) and the system error across the dataset. The normalization improves convergence speed by ensuring that the scale of the input features and outputs are even.

X, Y tuples provide a supervised learning framework to train a mapping from design parameters to system error. The dataset, consisting of 1567 samples, was split into three subsets to facilitate training, validation, and testing:

- Training: Contains 1085 samples. Used to train the model by updating its weights to minimize the loss function over the input-output mappings
- Validation: Contains 233 samples. Used during training to evaluate the model's performance on unseen data and to tune hyperparameters.
- Testing: Contains 232 samples. Used for final evaluation on unseen data after training is complete.

Our dataset consisted of 1567 data points. Each data point consists X, Y tuples. X is a 12-dimensional vector corresponding to the values of the 12 design parameters, $x_1, x_2, \ldots, x_{12}$. Y is a seven-dimensional vector, where the first six elements correspond to the six system-level FRs calculated for the X and the seventh element is the system error. of the exact reward values corresponding to a given value for the 12 design parameters. -output

The model consists of an input layer, a hidden layers, and an output layer:

- Input layer consists of 12 neurons. Input to the neural network consists of the 12 design parameters, $x_1, x_2, \ldots, x_{12}$.
- Hidden layer is a fully connected dense layer with 12 neurons. A hyperbolic tangent activation function is used for each neuron in the hidden layer.
- Output layer is a dense layer with 7 neurons and a hyperbolic tangent activation function. Tanh activation functions were used since the outputs are normalized in the range -1 to 1.

5.1.2 Surrogate Reward Model Results

The neural network was trained for 10000 epochs to provide sufficient iterations for convergence. The batch size was chosen to be 500 to maintain the right balance between training

speed and stability. The mean absolute error between the predicted and actual outputs on the training data was used as the loss function. The network parameters were trained using Adam's optimizer. The training loss and validation loss for 10000 epochs are plotted in Figure 5.1. At epoch 500, the training loss was 0.0736 and the validation loss was slightly higher at 0.0777. At epoch 10,000, the training loss further decreased to 0.0269 and the validation loss was also reduced to 0.0293. The neural network had successfully learned to minimize error and generalized very well to the validation set.

After training the model for 10,000 epochs, its performance was evaluated on the out-of-sample test dataset. The test loss was 0.0305, indicating the mean absolute error between the model's predictions and the actual target values in the test set. Thus, the model could generalize effectively to unseen data.

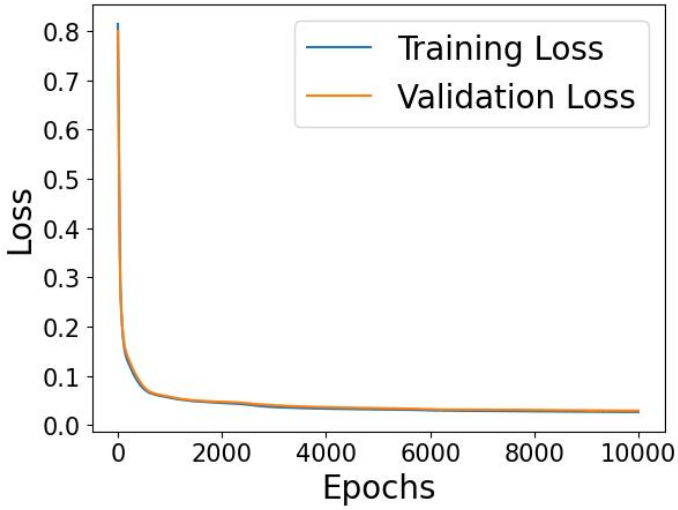


Figure 5.1: Training and validation loss for surrogate reward model.

.

Having designed and trained this neural network, one can benefit from RL's ability to iteratively refine the design by learning from feedback obtained based on this network. The RL process operates by associating a reward mechanism with each design iteration, encouraging the system to discover solutions that maximize performance according to predefined criteria. The surrogate reward predicted by the neural network in the previous section was used to model the reward function in RL. Next section describes the reinforcement learning environment.

5.2 Reinforcement Learning Environment

State space: The state space was modeled as a 12-dimensional space, with each dimension corresponding to one of the twelve design parameters. The bounds of each dimension were defined by the upper and lower limits of the respective design parameter.
Starting state: The state space is initialized with default parameters, which is the same as the parameters of the baseline design. These parameters are detailed in Table 3.4.

Action space: The action space was discrete and comprised two components. Component 1 represented the index of the design parameter to modify and could take twelve possible values, corresponding to the twelve design parameters. Component 2 specified how to modify the selected parameter, with three possible values: -1, 0, and 1. A value of -1 indicated decreasing the parameter value, 0 indicated no change, and 1 indicated increasing the parameter value. The parameter was adjusted (increased for +1 or decreased for -1) by the increment value specified in Table 3.2.

Reward function: We formulated the reward function based on the design quality. Every time the agent makes an action A^t at time step t it changes the design state from S^t to S^{t+1}. Let E^t and E^{t+1} denotes the normalized error associated with the states S^t and S^{t+1}, respectively. If the step resulted in a improved design quality (i.e., $E^{t+1} < E^t$) a positive reward is assigned to incentivize policies that enhance design quality. Conversely, If the step resulted in a diminished or unchanged design quality, i.e., $E^{t+1} \geq E^t$ then a penalty proportional to the degradation is applied. The penalty discourages the agent from taking actions that worsen the design. Mathematically, the reward function at time t is defined as:

$$R^t = \begin{cases} e^{-\left(E^t\sqrt{2\pi}\right)}, & \text{if } E^{t+1} < E^t \\ -0.9\left(E^{t+1} - E^t\right), & \text{if } E^{t+1} \geq E^t \end{cases} \tag{5.1}$$

E^t is estimated using the surrogate reward neural network model described in Section 5.1. This approach is necessary because reinforcement learning training involves thousands of reward calculations. Evaluating each state using finite element analysis would significantly increase training time. Therefore, a surrogate model based on the trained neural network is employed to expedite the process. The computed reward reflects the quality of the current

design and guides the agent towards improved designs. The reward function provides feedback to the agent about how well its actions contribute to optimizing the design, guiding it towards the best possible configurations while discouraging suboptimal changes.

Step: A step is a single interaction between the agent and the environment. A step at time t transitions the state from S^t to S^{t+1}, resulting in a reward R^t.

Episode: An episode is a sequence of steps that begins when the agent's state is initialized to baseline design parameters and continues as the agent takes actions, receives feedback or rewards, and transitions to new states. An episode terminates when one of the following conditions is satisfied:

- Reward R^t reaches a specified threshold of 1. A reward of 1 corresponds to a design with a normalized error of 0. Since only a design that satisfies all functional requirements has a normalized error of 0, the threshold for episode termination is set to 1.
- A pre-defined maximum number of steps is reached.

At the end of an episode, the agent begins a new episode with the state space reset to the baseline design.

The following two sections present the results of training a reinforcement learning agent using PPO and A2C within the environment outlined in this section.

5.3 PPO

5.3.1 PPO Hyper-parameters

PPO is a policy-gradient-based reinforcement learning algorithm that optimizes a policy by maximizing a surrogate objective while ensuring updates are constrained. PPO uses clipped objective functions and entropy regularization to encourage exploration and prevent drastic updates. PPO offers several advantages that make it well-suited for this task. The algorithm employs a clipped surrogate objective function, which prevents abrupt policy changes that could destabilize the learning process. PPO facilitates gradual and steady

policy adjustments, ensuring that the agent progresses without overlooking valuable design configurations.

The policy architecture is a Multilayer Perceptron, which means the agent uses a fully connected neural network to process the environment's state and decide actions. Multilayer Perceptron architecture is suitable for environments with low-dimensional state spaces, which was the case in the aircraft design task. Policy architecture network determines the best actions to take in each state. The hyperparameters used for PPO are tabulated in Table 5.1.

Hyperparameter	**Value**	**Description**
policy	MlpPolicy	multi-layer perceptron policy
learning rate	3e-4	step size for optimizer
number of steps	32	number of steps per environment update
batch size	16	number of samples used for training the policy
number of epochs	10	number of policy update epochs
gamma	0.99	discount factor for future rewards
vf_coef	0.5	coefficient for value function loss
clip_range	0.2	clipping parameter for policy loss
verbose	0	verbosity level (0: none, 1: info, 2: debug)
max number of steps	400	maximum per episode

Table 5.1: PPO Hyperparameter Settings

Some points we need to consider:

- The number of steps to collect in each rollout (trajectory) before performing an update was set to 32. Smaller values can lead to faster updates but less stable training, while larger values allow the model to better approximate gradients.
- Batch size, which is the number of samples used for training the policy in each update step, was set to 16. The batch size must be smaller or equal to number of steps.

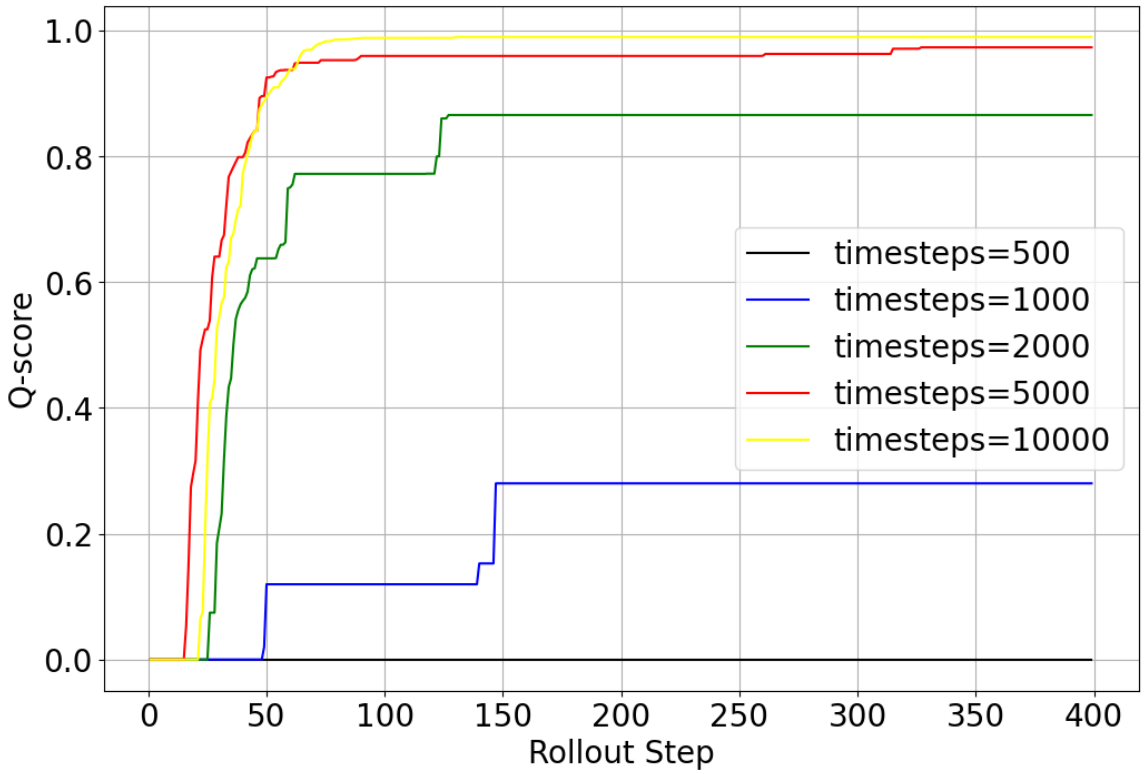


Figure 5.2: Policy rollout using PPO algorithm .

5.3.2 PPO Results

Figure 5.2 presents the rollout results of the PPO-trained policy. Each episode comprises 400 timesteps, with the figure depicting the Q-score associated with the design state attained at each timestep. The initial state corresponds to the baseline design, characterized by a Q-score of zero. From this starting point, the agent transitions through various design states by executing actions determined by the PPO-trained policy. The Q-scores of the resulting states are recorded and plotted, providing insight into the policy's performance in navigating the design space.

Five PPO policies were trained with varying training durations: 500, 1,000, 2,000, 5,000, and 10,000 timesteps. The black curve represents the policy rollout for the model trained with 500 timesteps, where the Q-score remained at zero throughout. This indicates that the PPO policy trained with 500 timesteps was unable to surpass the baseline design, consistently producing designs of lower quality.

For all policies trained with timesteps greater than 500, the resulting Q-scores exceeded zero, demonstrating improvements over the baseline design. The blue curve represents

the rollout for the policy trained with 1,000 timesteps, achieving a maximum Q-score of 0.28. The green curve corresponds to the policy trained for 2,000 timesteps, which reached a maximum Q-score of 0.87. The red curve illustrates the rollout for the policy trained over 5,000 timesteps, achieving a peak Q-score of 0.96. Finally, the yellow curve shows the rollout after training for 10,000 timesteps, yielding the highest Q-score of 0.99.

However, this improvement comes with a proportional increase in the number of reward evaluations required. Table 5.2 summarizes the maximum Q-scores achieved and the corresponding reward evaluation counts for each policy. It is important to note that each reward evaluation represents a computationally expensive model execution, highlighting the trade-off between improved design quality and computational cost.

Table 5.2: Q-scores and reward evaluation counts for PPO policies trained with varying timesteps.

Timesteps	Max Q-score	Reward evaluations
500	0	512
1000	0.28	1024
2000	0.87	2016
5000	0.96	5024
10000	0.99	10016

5.4 A2C

5.4.1 A2C Hyperparameters

The policy architecture for A2C is implemented as a Multilayer Perceptron, similar to PPO. The hyperparameters for A2C are presented in Table 5.3.

The number of steps per update is set to 5. In comparison, PPO had 32 steps per update. A2C typically uses fewer steps per update compared to PPO due to differences in how the two algorithms process data and manage policy updates. A2C uses the advantage function to reduce variance in policy gradient estimation. However, it lacks the clipping mechanism of PPO, so frequent updates with smaller steps help avoid instability due to overly large

Hyperparameter	Value	Description
policy	MlpPolicy	Multi-layer perceptron policy
learning rate	7e-4	Step size for optimizer
number of steps	5	Number of steps per environment update
batch size	16	Number of samples used for training the policy
gamma	0.99	Discount factor for future rewards
vf_coef	0.5	Coefficient for value function loss
entropy_coef	0.0	Coefficient for entropy term
verbose	0	Verbosity level (0: none, 1: info, 2: debug)
max number of steps	100	Maximum per episode

Table 5.3: A2C Hyperparameter Settings

policy updates. PPO's clipping mechanism allows it to safely handle larger updates, which can be derived from larger batches of data collected over more timesteps.

5.4.2 A2C Results

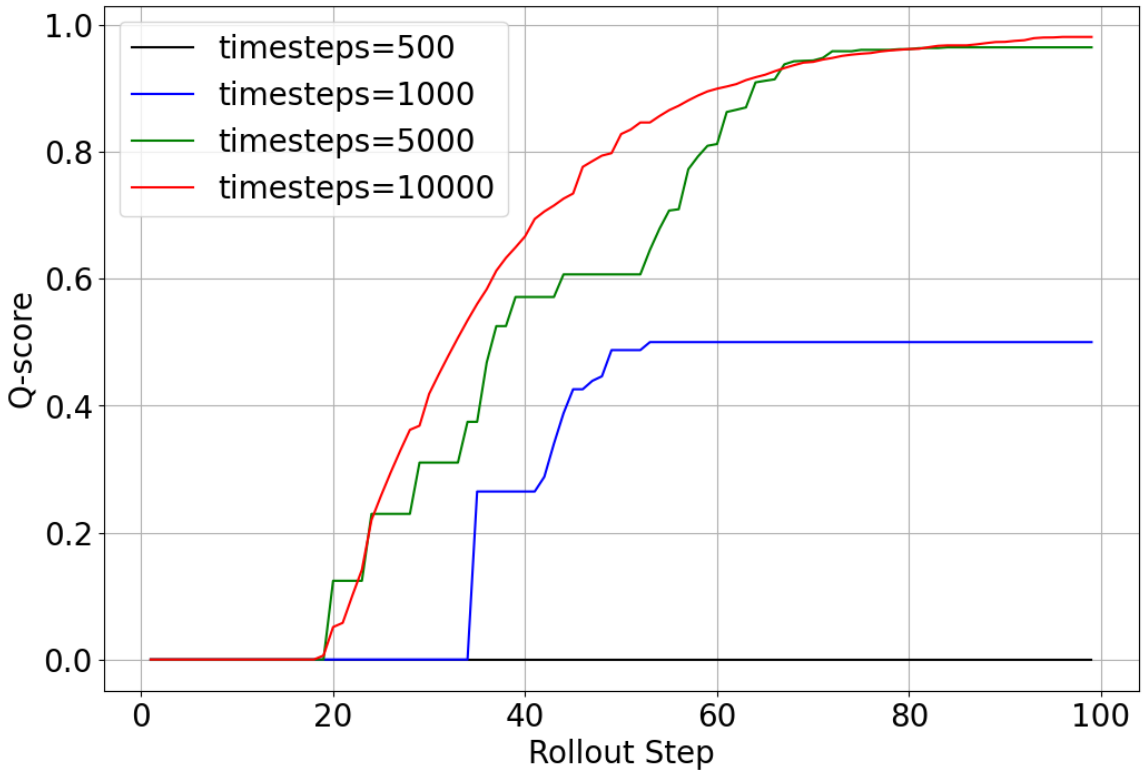


Figure 5.3: Policy rollout using A2C algorithm .

Figure 5.3 illustrates the performance of the A2C-trained policy. Similar to PPO, the initial state corresponds to the baseline design, characterized by a Q-score of zero. The agent transitions through design states by executing actions guided by the A2C-trained policy, and the resulting Q-scores are plotted to evaluate the policy's effectiveness in navigating the design space.

A2C policies were trained with timesteps of 500, 1,000, 5,000, and 10,000. The rollout for the policy trained with 500 timesteps (black curve) shows no improvement over the baseline, as the Q-score remained at zero throughout the episode, consistent with the observations from PPO for short training durations.

For policies trained with 1,000 or more timesteps, A2C demonstrated improvements over baseline design. The policy trained with 1,000 timesteps (blue curve) achieved a maximum Q-score of 0.5, higher than the equivalent PPO policy for this duration. The policy trained for 5,000 timesteps (green curve) reached a Q-score of 0.96, closely aligning with the PPO result. Finally, the rollout for the policy trained with 10,000 timesteps (red curve) achieved a peak Q-score of 0.98, slightly lower than PPO's best performance.

These results highlight that while A2C can achieve competitive performance, it requires comparable training durations to match the policy refinement observed with PPO. The differences in Q-scores across training durations also emphasize the sensitivity of each algorithm to the underlying design task and environment dynamics.

Table 5.4: Q-scores and reward evaluation counts for A2C policies trained with varying timesteps.

Timesteps	Max Q-score	Reward evaluations
500	0	512
1000	0.5	1024
5000	0.96	5024
10000	0.99	10016

Chapter 6

Discussion and Future Works

This chapter presents the key findings and insights gained from applying reinforcement learning (PPO and A2C) and imitation learning approaches to the aircraft design task. Key conclusions, limitations, and opportunities for future research are highlighted in this chapter.

6.1 Conclusion and Discussions

6.1.1 Imitation Learning Approach

We studied different recurrent neural network architectures—GRU, LSTM, and simple-RNN—using mean absolute error as a metric for assessing model accuracy. The results indicate that GRU outperforms both LSTM and simple-RNN in terms of raw accuracy, highlighting its superior performance in predicting design decisions in the context of the aircraft design task. The optimal performance was achieved with a lookback of 10, which refers to the number of previous design iterations the model considers when making predictions. For the given design task, a lookback of 10 strikes an ideal balance between capturing enough temporal context and avoiding overfitting to irrelevant past states.

GRU's superior performance over LSTM can be attributed to the specific nature of the design task, where long-term dependencies across a vast number of time steps are not critical for effective decision-making. The absence of the need for complex long-term

memory management allows GRU to achieve better generalization and avoid overfitting, especially in tasks with limited training data.

While GRU demonstrated faster training speeds compared to LSTM, the time difference between the two models was not significant for the small dataset used in this study, which consisted of a few thousand labeled samples. For datasets of this size, the computational efficiency offered by GRU did not yield a major advantage, as both models trained within similar time frames. However, for larger datasets with more complex tasks requiring extensive training, the reduced number of parameters and faster convergence of GRU would provide a more tangible benefit. Therefore, in the context of our study, GRU's balance of accuracy, training efficiency, and simplicity made it the most effective choice for the design optimization task.

The imitation learning agent, trained through behavioral cloning using human design decision data, demonstrated its ability to explore and navigate the design space for the aircraft design process. Beginning with a baseline design, the agent successfully improved the design, achieving a Q-score of 0.8 after approximately 150 iterations, progressing from the initial baseline configuration.

However, when trained with a reduced amount of human labels—approximately half the number used in the first training scenario—the agent achieved a much lower Q-score of 0.35. This significant decrease in performance suggests that the amount of human-labeled data plays a crucial role in teaching the agent the underlying patterns of successful design decisions. The agent relies on human design labels to transfer valuable design strategies and refine its decision-making process.

6.1.2 Reinforcement Learning Approach

Our study evaluated the effectiveness of two reinforcement learning approaches—PPO and A2C—for optimizing an aircraft design task. The results are summarized and compared based on Q-scores, training dynamics, and computational efficiency.

PPO achieved the highest Q-score of 0.99 after 10,000 timesteps. Its clipped surrogate objective and entropy regularization facilitated stable training and gradual policy refine-

ment. A2C, which relies on frequent updates with smaller step sizes, achieved a comparable maximum Q-score of 0.98 but outperformed PPO during early training stages, achieving a higher Q-score of 0.5 at 1,000 timesteps compared to PPO's 0.28. This highlights A2C's advantage in leveraging limited training durations effectively. However, as the training duration extended, PPO surpassed A2C, likely due to its ability to process larger data batches and avoid policy instability.

Both algorithms incurred high computational costs for reward evaluations, which increased proportionally with training duration, emphasizing the trade-off between design quality and computational efficiency. In comparison, imitation learning via behavioral cloning achieved a Q-score of 0.8, lower than the maximum Q-scores attained by PPO and A2C. The reliance of behavioral cloning on supervised human decision data limited its ability to explore and improve beyond the observations in the dataset. Despite its limitations in exploration and adaptability, behavioral cloning required significantly less computational time for training, as it bypassed the need for extensive reward evaluations. This makes it suitable for scenarios where computational resources or labeled data are constrained, such as engineering design.

In summary, while PPO and A2C demonstrated superior performance in achieving higher Q-scores through reward-driven exploration, imitation learning offered a faster, computationally efficient alternative at the cost of reduced exploration and adaptability.

6.2 Limitations

However, one limitation of our study is that it focuses on a single engineering design task, which may limit the generalizability of the findings to other design problems. Future research could investigate the application of transfer learning and meta-learning techniques to address this limitation. Transfer learning would involve training an agent on a related task and subsequently fine-tuning the agent on a new task with fewer samples, reducing the number of human-labeled data required for effective learning. Meta-learning, on the other hand, would aim to create a model that can quickly adapt to new tasks with minimal

data, further expediting the learning process. Such approaches could significantly reduce the reliance on large datasets and facilitate the application of agents to a wider range of engineering design tasks, ultimately enabling faster adaptation and learning with fewer labeled examples.

Another notable limitation of using behavioral cloning is that the RL agent's capacity for improvement is limited to the patterns observed within the provided human design decision data. In behavioral cloning, the agent learns by imitating the actions taken by humans based on labeled training data. However, this approach does not incorporate direct interaction with the environment's reward function, which means the agent does not engage in exploration beyond the observations provided in the training set. Consequently, the agent is unable to generate novel strategies or refine its policy through further exploration of the design space. This is because the agent is restricted to the decisions made by human designers in the training data, without feedback from an environment-driven reward signal that could guide it to improve its design decision-making abilities.

In reinforcement learning, the exploration-exploitation trade-off is a key factor that influences the agent's ability to discover new strategies and optimize its performance. However, reinforcement learning approaches are generally slower to converge. As a result, training such algorithms can be computationally expensive and time-consuming, particularly for complex design tasks.

6.3 Future Work

This research has demonstrated the potential of standalone reinforcement learning and imitation learning for automating complex engineering design tasks, such as aircraft design. A potential avenue for future research lies in developing advanced imitation learning techniques that combine the strengths of behavioral cloning with those of reinforcement learning. One such approach could involve integrating the human design decision data with environment dynamics and reward signals, creating a hybrid learning paradigm that benefits from both supervised learning and exploration. By incorporating a reward function

alongside human labels, the agent could not only mimic the behavior of human designers but also leverage the reward signal to guide exploration and optimization of the design space. This approach would allow the agent to explore beyond the observed design decisions, iteratively refining its policy to improve design quality and meet specific requirements more effectively. Incorporating these more advanced imitation learning algorithms could enable the development of RL agents that are capable of efficiently navigating complex design spaces.

There are a few more areas remain open for further exploration:

- **Generalization Across Design Domains**: Future research could extend the hybrid IL-RL framework to other engineering domains, such as automotive, architecture, or renewable energy systems. Investigating the adaptability of the framework across disciplines will reveal its robustness and potential for generalization. Techniques such as meta-imitation learning, inter-task learning, and transfer learning will be a potential area to explore to advance research in this direction.

- **Integration of Real-Time Feedback Mechanisms**: Incorporating real-time feedback from sensors or simulations during the learning process could enable the framework to address dynamic design requirements and environmental uncertainties. This advancement would enhance the applicability of the methodology in real-world engineering scenarios.

- **Exploration of Multi-Agent Collaboration**: Building on the collaborative aspects of engineering design, future studies could explore multi-agent systems where multiple RL and IL agents work together. This setup could simulate more realistic scenarios with interdependent subsystems, fostering deeper insights into distributed decision-making and optimization.

- **Augmented Reality (AR) and Virtual Reality (VR) Integration**: The integration of AR/VR tools into the design workflow could offer immersive and interactive visualization, enabling designers to interact with the AI-generated designs in a more intuitive

manner. This approach could improve designer-AI collaboration and accelerate the iterative process.

- **Scalability to Large-Scale Engineering Problems**: Future work should address the scalability of the proposed methods to larger-scale, industry-level design problems with significantly more variables and constraints. This extension will involve developing advanced decomposition strategies and hierarchical learning frameworks to manage complexity effectively.

Bibliography

Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004. doi: 10.1145/1015330.1015430.

Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009. doi: 10.1016/j.robot.2008.10.024.

Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017. doi: 10.1109/MSP.2017.2743240.

Autodesk. Model-based definition: Digitize all product engineering data. URL https://www.autodesk.com/solutions/model-based-definition.

SHAPOUR AZARM. Engineering design, 4th edition. *Journal of Mechanical Design*, 131(5): 056501, 04 2009. ISSN 1050-0472. doi: 10.1115/1.3116263.

D. Bertsekas. *Dynamic Programming and Optimal Control: Volume II; Approximate Dynamic Programming*. Athena Scientific optimization and computation series. Athena Scientific, 2012. ISBN 9781886529441.

Ghazal Bozorgmehry Boozarjomehry and Joseph Thekinen. Teaching ai to design from humans: a comparison of behavioral cloning architectures. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*,

volume 87318, page V03BT03A070. American Society of Mechanical Engineers, 2023. doi: 10.1115/DETC2023-115280.

Nicolas Bougie, Takashi Onishi, and Yoshimasa Tsuruoka. Interpretable imitation learning with symbolic rewards. *ACM Transactions on Intelligent Systems and Technology*, 15(1): 1–34, December 2023. ISSN 2157-6912. doi: 10.1145/3627822.

Yann Bouteiller. *Managing the World Complexity: From Linear Regression to Deep Learning*, pages 441–472. Springer Nature Singapore, Singapore, 2022. ISBN 978-981-19-1983-1. doi: 10.1007/978-981-19-1983-1_15.

Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc Le. Massive exploration of neural machine translation architectures, 2017. URL https://arxiv.org/abs/1703.03906.

Roberto Cahuantzi, Xinye Chen, and Stefan Güttel. A comparison of lstm and gru networks for learning symbolic sequences. In Kohei Arai, editor, *Intelligent Computing*, pages 771–785, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-37963-5.

LiChun Cao and ZhiMin. An overview of deep reinforcement learning. In *Proceedings of the 2019 4th International Conference on Automation, Control and Robotics Engineering*, CACRE2019, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450371865. doi: 10.1145/3351917.3351989.

Siva R. Chavali, Chiradeep Sen, Gregory M. Mocko, and Joshua D. Summers. Using rule based design in engineer to order industry: An sme case study. *Computer-Aided Design and Applications*, 5(1–4):178–193, January 2008. ISSN 1686-4360. doi: 10.3722/cadaps.2008.178-193.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches, 2014a. URL https://arxiv.org/abs/1409.1259.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-

decoder for statistical machine translation, 2014b. URL https://arxiv.org/abs/1406.1078.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014. URL https://arxiv.org/abs/1412.3555.

N. Cross. *Engineering Design Methods: Strategies for Product Design*. Wiley, 2021. ISBN 9781119724407.

Rachit Dubey, Pulkit Agrawal, Deepak Pathak, Thomas L. Griffiths, and Alexei A. Efros. Investigating human priors for playing video games, 2018. URL https://doi.org/10.48550/arXiv.1802.10217.

Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning, 2019. URL https://arxiv.org/abs/1904.12901.

Steven D. Eppinger and Tyson R. Browning. *Design Structure Matrix Methods and Applications*. The MIT Press, May 2012. ISBN 9780262301428. doi: 10.7551/mitpress/8896.001.0001.

Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 12(10):2451–2471, October 2000. ISSN 1530-888X. doi: 10.1162/089976600300015015.

Wonjoon Goo and Scott Niekum. Know your boundaries: The advantage of explicit behavior cloning in offline RL, 2023. URL https://doi.org/10.48550/arXiv.2206.00695.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. ISBN 9780262035613.

Alex Graves. Generating sequences with recurrent neural networks, 2014. URL https://arxiv.org/abs/1308.0850.

Klaus Greff, Rupesh K. Srivastava, Jan Koutnik, Bas R. Steunebrink, and Jurgen Schmidhuber. Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning*

Systems, 28(10):2222–2232, October 2017. ISSN 2162-2388. doi: 10.1109/tnnls.2016.2582924.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016a. URL https://doi.org/10.48550/arXiv.1606.03476.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning, 2016b. URL https://arxiv.org/abs/1606.03476.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, November 1997. ISSN 1530-888X. doi: 10.1162/neco.1997.9.8.1735.

Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017. doi: 10.1145/3054912.

Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

Michael Kelly, Chelsea Sidrane, Katherine Driggs-Campbell, and Mykel J Kochenderfer. Hg-dagger: Interactive imitation learning with human experts. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8077–8083. IEEE, 2019.

H. Kerzner. *Project Management: A Systems Approach to Planning, Scheduling, and Controlling*. Wiley, 2017. ISBN 9781119165354.

Xiaojin Li, Yueyang Huang, and Yuanbo Shi. Ultra-short term power load prediction based on gated cycle neural network and xgboost models. *Journal of Physics: Conference Series*, 2026 (1):012022, September 2021. ISSN 1742-6596. doi: 10.1088/1742-6596/2026/1/012022.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019. URL https://arxiv.org/abs/1509.02971.

Zachary C. Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning, 2015. URL https://arxiv.org/abs/1506.00019.

Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. An intriguing failing of convolutional neural networks and the coordconv solution. *Advances in neural information processing systems*, 31, 2018.

Andrew G Lowe and Nathan W Hartman. A case study in cad design automation. *Journal of Technology Studies*, 37(1):2–9, 2011. doi: 10.21061/jots.v37i1.a.1.

Trent W Lukaczyk, Andrew D Wendorff, Michael Colonno, Thomas D Economon, Juan J Alonso, Tarik H Orra, and Carlos Ilario. SUAVE: an open-source environment for multi-fidelity conceptual vehicle design. In *16th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, number AIAA-2015-3087, 2015. doi: 10.2514/6.2015-3087.

Timothy MacDonald, Matthew Clarke, Emilio M Botero, Julius M Vegh, and Juan J Alonso. SUAVE: an open-source environment enabling multi-fidelity vehicle optimization. In *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, number AIAA-2017-4437, 2017. doi: 10.2514/6.2017-4437.

Galina Medyna, Sarayut Nonsiri, Eric Coatanéa, and Alain Bernardb. Modelling, evaluation and simulation during the early design stages: Toward the development of an approach limiting the need for specific knowledge. *Journal of Integrated Design amp; Process Science*, 16(3):111–131, 2012. ISSN 1092-0617. doi: 10.3233/jid-2012-0006.

Simon W Miller, Michael A Yukish, and Timothy W Simpson. Design as a sequential decision process: A method for reducing design set space using models to bound objectives. *Structural and Multidisciplinary Optimization*, 57:305–324, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Kirkeby Fidjeland, Georg Ostro-

vski, Stig Petersen, Charlie Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015. URL https://api.semanticscholar.org/CorpusID:205242740.

Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Bridging the gap between value and policy based reinforcement learning. 2017a. URL https://arxiv.org/pdf/1702.08892.pdf.

Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Bridging the gap between value and policy based reinforcement learning. *Advances in neural information processing systems*, 30, 2017b.

Andrew Y Ng, Stuart Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

P. O'Connor and A. Kleyner. *Practical Reliability Engineering*. Quality and Reliability Engineering Series. Wiley, 2012. ISBN 9780470979815.

Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J. Andrew Bagnell, Pieter Abbeel, and Jan Peters. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1–2):1–179, 2018. ISSN 1935-8261. doi: 10.1561/2300000053. URL http://dx.doi.org/10.1561/2300000053.

Gerhard Pahl, Wolfgang Beitz, Jörg Feldhusen, and Karl-Heinrich Grote. *Engineering Design*. Springer London, 2007. ISBN 9781846283192. doi: 10.1007/978-1-84628-319-2. URL http://dx.doi.org/10.1007/978-1-84628-319-2.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.

A.T.D. Perera and Parameswaran Kamalaruban. Applications of reinforcement learning in energy systems. *Renewable and Sustainable Energy Reviews*, 137:110618, 2021. ISSN 1364-0321. doi: 10.1016/j.rser.2020.110618.

Raphaël Pestourie, Youssef Mroueh, Chris Rackauckas, Payel Das, and Steven G. Johnson. Physics-enhanced deep surrogates for partial differential equations. *Nature Machine Intelligence*, 5(12):1458–1465, December 2023. ISSN 2522-5839. doi: 10.1038/s42256-023-00761-y.

Dean A. Pomerleau. Alvinn: an autonomous land vehicle in a neural network. In *Proceedings of the 1st International Conference on Neural Information Processing Systems*, NIPS'88, page 305–313, Cambridge, MA, USA, 1988. MIT Press.

Warren B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality (Wiley Series in Probability and Statistics)*. Wiley-Interscience, USA, 2007. ISBN 0470171553.

Itziar Ricondo, Alain Porto, and Miriam Ugarte. A digital twin framework for the simulation and optimization of production systems. *Procedia CIRP*, 104:762–767, 2021. ISSN 2212-8271. doi: 10.1016/j.procir.2021.11.128.

Jana I. Saadi and Maria C. Yang. Generative Design: Reframing the Role of the Designer in Early-Stage Design Process. *Journal of Mechanical Design*, 145(4):041411, 02 2023. ISSN 1050-0472. doi: 10.1115/1.4056799.

M.H. Sadraey, P. Belobaba, J. Cooper, R. Langton, and A. Seabridge. *Aircraft Design: A Systems Engineering Approach*. Aerospace Series. Wiley, 2012. ISBN 9781118352809.

Amir Sahraei, Tobias Houska, and Lutz Breuer. Deep learning for isotope hydrology: The application of long short-term memory to estimate high temporal resolution of the stable isotope concentrations in stream and groundwater. *Frontiers in Water*, 3, September 2021. ISSN 2624-9375. doi: 10.3389/frwa.2021.740044.

Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *arXiv preprint arXiv:1704.02532*, 2017.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

David Silver and Joel Veness. Monte-carlo planning in large pomdps. *Advances in neural information processing systems*, 23, 2010.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–, October 2017. URL http://dx.doi.org/10.1038/nature24270.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013. doi: 10.18653/v1/d13-1170.

Wen Sun, Arun Venkatraman, Geoffrey J. Gordon, Byron Boots, and J. Andrew Bagnell. Deeply aggrevated: Differentiable imitation learning for sequential prediction, 2017. URL https://arxiv.org/abs/1703.01030.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

J Thekinen and P Grogan. Information exchange patterns in digital engineering: An observational study using web-based virtual design studio. *Journal of Computing and Information Science in Engineering*, 21(4):041012, 2021. doi: 10.1115/1.4050087.

Joseph Thekinen and Paul T Grogan. Effects of augmented information system on design communication: A human-subject study using aircraft design studio. In *International Design Engineering Technical Conferences and Computers and Information in Engineering*

Conference, volume 86212, page V002T02A064. American Society of Mechanical Engineers, 2022. doi: 10.1115/DETC2022-91086.

Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*, 2018.

D.G. Ullman. *The Mechanical Design Process*. McGraw-Hill series in mechanical engineering. McGraw-Hill Higher Education, 2010. ISBN 9780071267960.

Greg Van Houdt, Carlos Mosquera, and Gonzalo Nápoles. A review on the long short-term memory model. *Artificial Intelligence Review*, 53(8):5929–5955, May 2020. ISSN 1573-7462. doi: 10.1007/s10462-020-09838-1.

Jozef Vaskỳ, Michal Eliáš, Pavol Bezák, Zuzana Červeňanská, and Ladislav Izakovič. 3d model generation from the engineering drawing. *Research Papers Faculty of Materials Science and Technology Slovak University of Technology*, 18(29), 2010. ISSN 1336-1589. doi: 10.2478/v10186-010-0025-z.

Xu Wang, Sen Wang, Xingxing Liang, Dawei Zhao, Jincai Huang, Xin Xu, Bin Dai, and Qiguang Miao. Deep reinforcement learning: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 35(4):5064–5078, 2024. doi: 10.1109/TNNLS.2022.3207346.

Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 737–744, 2020. doi: 10.1109/SSCI47803.2020.9308468.