IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 4 December 2023, accepted 25 January 2024, date of publication 29 January 2024, date of current version 5 February 2024.

*Digital Object Identifier 10.1109/ACCESS.2024.3359417*

RESEARCH ARTICLE

# Optimal Reusable Rocket Landing Guidance: A Cutting-Edge Approach Integrating Scientific Machine Learning and Enhanced Neural Networks

**UGURCAN ÇELIK**[1] **AND MUSTAFA UMUT DEMIREZEN**[2], **(Senior Member, IEEE)**
[1]Centre for Cyberphysical and Autonomous Systems, Cranfield University, MK43 0AL Bedford, U.K.
[2]Data Products Department, UDemy Inc., San Francisco, CA 94107, USA

Corresponding author: Mustafa Umut Demirezen (umut@demirezen.tech)

**ABSTRACT** This study presents an innovative approach that utilizes scientific machine learning and two types of enhanced neural networks for modeling a parametric guidance algorithm within the framework of ordinary differential equations to optimize the landing phase of reusable rockets. Our approach addresses various challenges, such as reducing prediction uncertainty, minimizing the need for extensive training data, improving convergence speed, decreasing computational complexity, and enhancing prediction accuracy for unseen data. We developed two distinct enhanced neural network architectures to achieve these objectives: Adaptive (AQResNet) and Rowdy Adaptive (RAQResNet) Quadratic Residual Neural Networks. These architectures exhibited outstanding performance in our simulations. Notably, the RAQResNet model achieved a validation loss approximately 300 times lower than the standard architecture with an equal number of trainable parameters and 50 times lower than the standard architecture with twice the number of trainable parameters. Furthermore, these models require significantly less computational power, enabling real-time computation on modern flight hardware. The inference times of our proposed models were measured in approximately microseconds on a single-board computer. Additionally, we conducted an extensive Monte Carlo analysis that considers a wide range of factors, extending beyond aerodynamic uncertainty, to assess the robustness of our models. The results demonstrate the impressive adaptability of our proposed guidance policy to new conditions and distributions outside the training domain. Overall, this study makes a substantial contribution to the field of reusable rocket landing guidance and establishes a foundation for future advancements.

**INDEX TERMS** Adaptive activation functions, guidance, navigation, optimal control, scientific machine learning, quadratic residual neural networks.

## I. INTRODUCTION

The development of reusable vertical take-off and vertical-landing (VTVL) technology has long been a hot topic in the aerospace industry. Recent interest in the space industry has centered on the capability of rockets to perform autonomously accurate vertical landings, allowing them to be reused for future missions [1]. This is due to a significant reduction in mission launching costs and an improvement in mission responsiveness. Launching a rocket is an expensive and risky endeavor, and there is a growing interest in how to make it more cost-effective. Constructing VTVL rockets is an efficient approach to reducing expenses and generating corporate value. The achievement of triumphant retrieval of the reusable stage requires the creation of a guidance command in real-time, which reduces the usage of propellant while adhering to numerous limitations imposed during several stages of the flight, ranging from reentry to landing [2].

The associate editor coordinating the review of this manuscript and approving it for publication was Alba Amato.

The application of traditional nonlinear optimization techniques is deemed inappropriate for real-time trajectory optimization of a reusable launch vehicle owing to its susceptibility to initial guesses and prolonged convergence time. The possibility of utilizing real-time, online trajectory optimization for reusable launch vehicle guidance is increasing owing to advancements in onboard computing power and optimization methods that are more efficient. Convex optimization is a frequently employed optimization technique in the aerospace engineering domain owing to its superior computational efficiency, as evidenced by multiple studies [3].

Guidance planning exhibits variability across distinct phases of the return mission, owing to the diverse objectives and dynamics encountered in each phase. The rocket recovery mission is often separated into four flying phases: the attitude management stage, the power reduction stage, the aerodynamic slowing stage, and the vertical landing stage so that the rocket falls gently and vertically at the designated site. The objective of the landing guidance system during the landing phase is to direct the booster towards the intended landing location, which may either be the launch site or a barge, contingent upon the reentry scenario and ensure a precise landing [2]. The guidance system must effectively manage the rapid changes during the return stage, especially when performing the powered landing stage, and must exhibit resilience in the face of uncertainties related to the vehicle and environment, such as wind conditions. The instructions necessitate the production of a reference trajectory and attitude directives that the control system will follow. Typically, the system operates in a closed-loop configuration, albeit at a low frequency, to separate it from the closed-loop dynamics of the control system [2]. The recovery process is hampered by obstacles such as vast swaths of the airspace and velocity domain, substantial changes in the flying environment, complicated flight limitations, and powerful perturbations and uncertainties, all of which may result in errors. The cumulative errors caused by the prior stages must be addressed during the final stage of the recovery operation, namely the vertical landing stage [4]. This delicate process must be completed with high performance and precision over a short period of time. Thus, highly stringent restrictions are imposed on the speed and precision of the guide during this phase. For a rocket to be successfully recovered, research on rapid and precise guiding methods during the vertical landing phase is essential.

### A. LITERATURE REVIEW

In studies [5], [6], authors utilized the traditional guidance approaches. The thrust profiles of the booster during landing are estimated using time-dependent polynomials, the coefficients of which are determined in order to fulfill the endpoint restrictions. However, these classical guidance algorithms often make unrealistic assumptions, such as neglecting aerodynamic models when designing guidance commands. Furthermore, these approaches lack an optimality criterion or an objective function. In real-world applications, fuel optimality is essential for reducing operational costs. By contrast, optimal guidance/control algorithms enable the definition of an objective function. Optimal control problems can be solved to generate fuel-optimal trajectories. However, the computational complexity of solving Optimal Control Problems (OCPs) makes on-board (online) implementation impractical [7], [8]. To overcome this challenge, extensive research has been conducted in the field of the convexification of OCPs, which allows on-board computations. In the second category, direct methods(optimal control approaches), [1], [9], [10], [11] have proposed the convexification of the optimal control problem (OCP) associated with vertical landing. The goal was to minimize the amount of fuel consumed during the mission. However, a robustness analysis, which includes the impacts of existing delays, sensor noise, and disturbances that might be encountered during landing, was not considered in these studies. Furthermore, while simplified low-fidelity real-time generation of optimal trajectories is feasible, achieving high-fidelity optimal trajectories remains a challenge [12]. Even in studies such as [13], where successive convexification of a 6-degree-of-freedom model for Mars landing was achieved, simplified model assumptions were made, such as using a constant inertia matrix during landing, which is unrealistic during fuel consumption. To address these computational limitations, [8] discussed the use of low-frequency guidance command generation with a high-fidelity 6-degree-of-freedom model. The authors employed a successive convexification methodology at the beginning of the powered landing phase and used a reference tracking controller to follow the trajectory throughout the landing. However, this approach was sensitive to disturbances encountered during the landing. Re-computing the optimal trajectory at each guidance loop is necessary to provide robustness to off-nominal flight conditions [12].

Recent research has indicated that in addition to the success achieved through the application of optimization theory-based guidance and control methods, there have been efforts to investigate the potential utilization of AI-based approaches in relation to the optimal guidance and operation of space/aerospace vehicle systems. The fundamental concept of this particular approach is to create an efficient guidance and control system using deep learning (DL) techniques [14]. In recent years, the rapid advancement of artificial intelligence research has enabled the achievement of optimum guidance and control in real-time [15]. The decrease in the computing complexity of network training and the new construction of NN models, in particular, has garnered a great deal of academic interest. The superior approximation and relational mapping capabilities of Deep Neural networks (DNNs) have spawned novel approaches for addressing guidance issues that compensate for the inaccuracy and lack of convergence of standard numerical techniques [16]. The utilization of DNN and recurrent neural networks (RNN) has been observed in the context of addressing reusable rocket landings and planetary landing problems. In [8], the authors proposed a real-time guidance policy called Deep Classification and Regression Network-based Guidance (DCRNG), which employs two DNNs to establish a nonlinear mapping between the ideal state and control pairings. This was achieved by fitting the optimal trajectory data generated by solving OCPs. Similarly, in previous studies [17], [18], [19], [20], DNNs were utilized to generate real-time guidance commands for mars powered descend landing problems. The authors of [21] proposed a guidance methodology for asteroid landing that utilizes DNNs to address the challenges related to convergence and real-time performance of solving optimal control problems with shooting methods. This approach aims to employ DNNs to provide an excellent initial estimate (solution) for the state and co-state trajectories to accelerate the convergence of the shooting method. As seen from recent literature, Deep learning (DL) may typically be implemented in guiding applications in two ways: direct and indirect methods. In the former approach, DNNs are trained to develop a nonlinear mapping between the optimal state and control pairings. Therefore, the trained NNs may steer a flight in real-time based on the observed/measured vehicle states [22]. In the latter approach, however, NN models often act as a supporting system to give "hot-start" initial values with the aim of accelerating real-time trajectory optimization methods. The authors opted for larger models in both approaches, employing deep learning techniques to fit discrete data precisely. This choice led to favorable performance metrics during training, and the trained parametric policies often accomplished the mission. However, the absence of robustness analysis against uncertainties poses a limitation. The model's performance degrades for cases beyond the distribution it was trained on. Given training in the discrete-time domain, minor errors originating from policy actions accumulate over time in continuous time, resulting in substantial errors at the final time.

Another AI-based approach is Reinforcement Learning (RL). This can be defined as a formalized approach to learning through trial and error. The fundamental premise of RL is that a machine can independently learn the optimal behavior or policy required to perform a specific task within a given environment. This is achieved by maximizing or minimizing the cumulative reward or cost. In [23] and [24], Reinforcement Learning based methods were proposed as a solution for the powered landing of reusable rockets. In RL, an agent can solve trajectory issues by learning a control strategy that maximizes the predicted reward. However, as the complexity of the task increases, the state space examined by the agent expands rapidly. Deep Reinforcement Learning (DRL) combines the concepts of DL and RL, representing an advanced form of RL that integrates deep neural networks. Hence, the most difficult aspect of DRL-based approaches is the search for global optimum solutions, such as fuel-optimal trajectories, and their training takes so much time.

### B. MOTIVATION AND CONTRIBUTIONS

Recent research on classical and learning-based optimum guiding algorithms has yielded encouraging findings. However, there are still challenges to overcome in the Reusable VTVL rocket landing problem. The primary motivation behind our study is to address the challenges encountered during the powered descent stage of recoverable rocket retrieval, aiming to achieve accurate and effective landings. The pragmatic viewpoint assumes significance in this particular context, given that the reutilization of rockets carries substantial significance with respect to curbing mission expenses, enhancing mission agility, and augmenting overall economic advantages. This research endeavors to devise a comprehensive guidance strategy that effectively addresses the aforementioned challenges and the gaps in the current literature and enhances the accuracy and dependability of landings of reusable rockets. The impetus behind our study is primarily rooted in the pragmatic necessity to attain accurate, streamlined, and dependable touchdown procedures for reusable rockets. By integrating optimal control and learning-based approaches to harness their individual strengths to overcome the limitations of non-learning and non-optimal methods and implementing cutting-edge methodologies, such as scientific machine learning and adaptive neural networks, we endeavored to overcome the obstacles faced by the aerospace sector, thereby facilitating its practical progress.

In this study, the use of scientific machine learning and adaptive quadratic neural networks with the universal ordinary differential equations framework is suggested as an innovative approach for the boosted landing phase of reusable rockets to overcome the problems and deficiencies mentioned in the literature. In this context, our contributions to the literature are summarized as follows:

1) To handle uncertainties, improve the generalization capability of NNs, and increase the out-of-domain performance of NN models, we propose the use of the universal ordinary differential equations (UODE) [25] approach in scientific machine learning [26], [27]. NNs can be trained not only on data but also on several physics laws using UDOE, combining data-driven machine-learning techniques with knowledge of physical principles and scientific models. Consequently, NN models can learn from data and physical laws to reduce uncertainty and improve extrapolation and out-of-domain prediction capabilities.
2) In general, optimal control problems require state-space models. From a machine-learning perspective, this is a regression task. NNs are universal function approximators with a certain degree of error. However, navigation and guidance algorithms must have precise outputs in order to control an aircraft. To use machine learning algorithms such as NNs, XGBoost, and Support Vector Regressors for navigation and guidance, the output predictions must be extremely accurate, easy to understand, and have little uncertainty.

The learning capacity of an NNs is directly related to the number of layers and neurons in these layers. Adding more of these hyper-parameters also requires more computing power, which limits the models that can be used in modern flight hardware. To improve the neural network capacity and prediction performance without increasing the network parameters, we propose two new types of neural networks based on Quadratic Residual Neural Networks (QResNet) [28] called Adaptive Quadratic Residual Neural Networks (AQResNet) and Rowdy Adaptive Quadratic Residual Neural Networks (RAQResNet) to speed up training, improve convergence, increase prediction accuracy [29], reduce spectral bias [30], [31], and reduce the size of the network parameters. We accomplished this by integrating adaptive activation functions [32], [33] into QResNet so that the NN could benefit from additional trainable parameters and provide it with a greater degree of freedom. We also suggested another type of QResNet called Rowdy Adaptive QResNet (RAQResNET), which uses special trigonometric functions [34] in the activation functions with particular arithmetic operations to smooth the training loss surface and prevent falling into local optima during network training.

3) We considered physics-informed neural networks [35] and constructed custom loss functions to train AQResNet and RAQResNet using the universal ordinary differential equations mathematical framework. We accomplish this by adding optimization constraints from the OCP to the loss function and using automatic differentiation to determine the gradients for all trainable network parameters. We trained these novel neural networks using computed gradients with gradient descent-based optimization methods.
4) Finally, we also tested our proposed models on a limited-resource single-board computer. Test results demonstrated inference times at the level of several microseconds, indicating no additional computational load on the guidance hardware system. The models' efficiency and improved learning capacity allow for better prediction accuracies with fewer layers and neurons. Rigorous testing confirmed that each guidance step could be executed within microseconds, underscoring the practicality and viability of real-time implementation.

Our study makes significant contributions from a pragmatic perspective and introduces a new approach to scientific machine learning in the aerospace industry by utilizing adaptive activation-based quadratic residual neural networks and integrating regression-based tasks into the universal ordinary differential equations framework. This approach improves the learning ability of the NNs, the rate of convergence, and the precision of predictions while taking into account the underlying physical principles and scientific models. In addition, our experimental results, which incorporate Monte Carlo analysis, demonstrate the resilience of our trained network models when confronted with disruptions and unpredictability. The models exhibited a notable capability to generalize effectively to unfamiliar conditions, thereby underscoring their practicality in real-world scenarios and their efficacy in acquiring knowledge of the underlying physical principles. This study underscores the practical implications of the research by emphasizing its potential applications. For instance, this approach can be implemented in high-fidelity simulations using realistic subsystem models. This highlights the adaptability of this research in the real world and its broader practical significance.

To the best of our knowledge, there has been no previous investigation within the aviation research field on the utilization of scientific machine learning in conjunction with universal ordinary differential equations (UODE) to solve reusable rocket landing problems. This study proposes and extensively experiments with such a groundbreaking guidance approach from an artificial intelligence perspective. Furthermore, this study introduces novel adaptive activation-based quadratic residual neural networks and provides comprehensive insights into leveraging regression-based tasks to address the challenges of highly nonlinear and complex function approximations. Integrating these newly proposed enhanced neural networks with UODE within the scientific machine learning framework is particularly notable, representing a significant contribution to the field. This exploration of integration advances research in this domain, paving the way for improved solutions and further improvements in rocket landing guidance.

The remainder of this paper is organized as follows: In Section II, the problem definition for the reusable rocket landing problem is introduced. In Section III, the methodology followed and the mathematical preliminaries are explained in detail, including the motion model derivation for rocket landing, optimal control problem definition, classical and scientific machine learning approaches in conjunction with the mathematical preliminaries, and formulations performed. Section IV introduces the simulations and shares the results for different scenarios, including out-of-domain trajectory performance, the robustness of uncertainty, and Monte Carlo analysis for six different conditions. Finally, in Sections V and VI, a summary of the outcomes of this study is presented.

## II. PROBLEM DEFINITION

The problem addressed in this study is the development of a parametric guidance command generator for reusable rocket recovery missions, which enables real-time computation. In the proposed approach, the developed policy directly maps the system states to the guidance commands for the controller loops. Considering the aforementioned challenges of the powered descent phase of the rocket engine in detail, it is necessary to achieve zero velocity at exactly zero altitudes with the upright attitude position to prevent the booster from tipping over; otherwise, the mission fails, and in fact, it might result in a catastrophic situation. Therefore, the developed guidance policy must bring the booster to the desired landing location with high precision. The second challenge concerns the uncertainties encountered during the landing phase or even before the flight. Environmental uncertainties, such as unexpectedly strong winds and gusts during landings, fall under the first category. Significant uncertainties at the system level can also arise in the second category. For instance, uncertainties in the aerodynamic coefficients, the position of the center of gravity, thrust magnitude level, actuator misalignment, and actuator bandwidth can be considered in this category. Finally, fuel efficiency is of considerable importance in the powered descent phase. The trajectories followed should be fuel-optimal trajectories. Otherwise, the booster can crash when running out of fuel before touching it down. That is, the second attempt can only be tried with additional fuel on board, which is rare in most situations [9].

Hence, the developed guidance algorithm in this study is expected to steer the booster to the desired landing location without any constraint violation. Simultaneously, it must address all challenging scenarios. In other words, robust, high-precision guidance that steers the booster in a fuel-optimal trajectory must be achieved.

$$U^{*}(X(t)) = \pi(X, p) = \arg\min_{\pi} m_{fuel} \tag{1}$$

As described in Equation 1, a parametric guidance policy is adopted to map the system states $X(t)$ to the feed-forward control inputs $U(t)$, $\pi : X(t) \rightarrow U(t)$. The main objective of the policy is to minimize fuel usage throughout the landing phase. In simple terms, the optimized policy should decide the action to take, based on the current states, to minimize fuel usage and satisfy the mission requirements. In our proposed approach, the developed policy maps the system states to the control inputs, which can be regarded as feed-forward guidance commands for the controller loops.

## III. MATERIALS AND METHODS

This study investigated the guidance policy development process using two approaches. The first approach concerns classical, fully data-driven machine-learning techniques. The second approach is the scientific machine learning technique, which can be regarded as a partially data-driven and physics-informed machine learning technique. The flowchart in Figure 1 summarizes the common steps followed by both approaches.

In the first step of the policy development process, we formulated the Optimal Control Problem (OCP), which included system dynamics, constraints, and boundary conditions for the booster landing mission, to obtain the train-test data needed for both aforementioned approaches. Then, the formulated OCPs are solved using RSOPT, an optimal control problem solver [36]. Subsequently, we stored the generated trajectories in the trajectory database for model training. In the model training part, several methods are employed for classical and scientific machine learning approaches to train a guidance policy. For classical machine learning, methods such as linear regression, XGBoost (a state-of-the-art gradient boosting decision tree algorithm) [37], and vanilla neural network architecture have been applied. On the other hand, the adaptive neural network architectures AQResNet and RAQResNet that we proposed are utilized for both classical and scientific machine learning approaches.

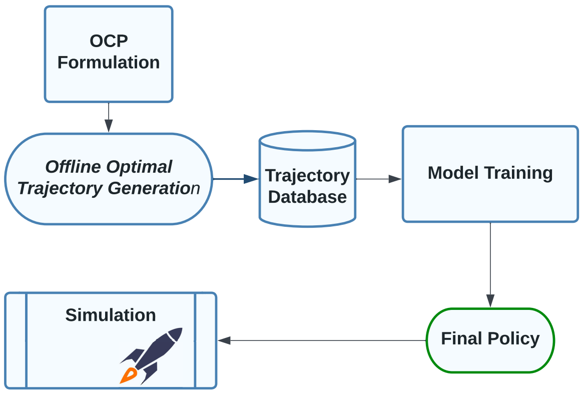

**FIGURE 1.** **General policy development flowchart.**

In the last step of the policy development framework, the performance of the optimized guidance policy was assessed through simulations. The evaluations were based on the mission requirements, including the touchdown location, velocity, attitude angle, fuel consumption metric, interpolation capabilities, and out-of-sample (extrapolation) dataset performances measured to understand the uncertainty reduction performance.

In this study, we made several assumptions for theoretical calculations:

1) The earth's surface is assumed to be flat and non-rotating.
2) The aerodynamic moment contributions were considered negligible compared with the moment produced by the thrust vector control.
3) The aerodynamic coefficients are only functions of the angle of attack, $C_D \approx C_D(\alpha)$ and $C_L \approx C_L(\alpha)$.
4) The drag coefficient was assumed to be a quadratic function of the angle of attack, whereas the lift coefficient had a linear relationship with the angle of attack.
5) The inner-control subfunction was not addressed in this work, and only the guidance subfunction was focused on.

All the proposed approaches and their mathematical foundations are given in the following sections:

### A. FLIGHT MECHANICS MODEL

In this section, a flight mechanics model for vertical landing vehicles is introduced. Because the time required for the boost landing phase is short, the Earth's surface is assumed to be flat and nonrotating. In addition to these assumptions, Having a 3-degree-of-freedom flight mechanics model was considered sufficient, including two transnational dynamics and one rotational dynamic.

### 1) REFERENCE FRAMES

The first reference frame is the landing location inertial reference frame (LLIF), as shown in Figure 2. In this reference frame, the origin $O$ is attached to the predefined landing location, the $OY$ axis is perpendicular to the plane, and the $OZ$ axis points downward. $OX$ axis follows the right-hand rule and points to the right. The second reference frame is the body-fixed reference frame attached to the booster's center of gravity. As shown in Figure 2, the $y$-axis of the body-fixed coordinate system is perpendicular to the plane, and the positive $x$-axis of the body-fixed coordinate system is towards the booster's nose. When pitch angle $\theta$ is defined as the angle between the x-axis of the body-fixed coordinate system and the x-axis of the LLIF,

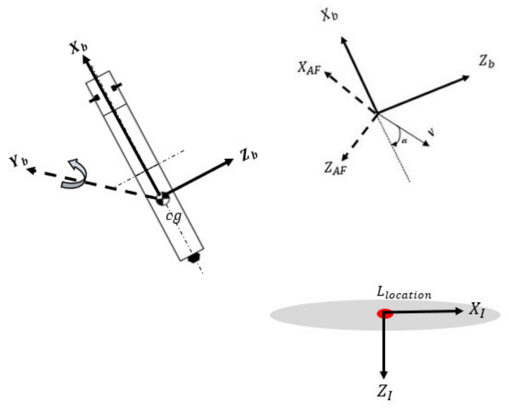

**FIGURE 2. Landing reference frame and body-fixed frame.**

In this study, the angle of attack of the landing vehicle is redefined as the effective angle of attack, which is the angle between the negative body x-axis and the velocity vector. Based on this definition, the aerodynamic forces acting on the landing vehicle are applied to the vehicle in the aerodynamic reference frame. The x-axis of this reference frame was aligned with the velocity vector, whereas the positive axis was aligned on the opposite side. The z-axis is perpendicular and positive in the downward direction. This reference frame is shown in the top-right corner of Figure 2.

### 2) DYNAMICS AND CONSTRAINTS

With the reference frame definitions given above, the equations of motion for the landing vehicle derived relative to the LLIF and resolved in the body-fixed coordinate system are given in Equation 2:

$$\begin{aligned}
\dot{x} &= u\cos\theta + w\sin\theta, \\
\dot{z} &= -u\sin\theta + w\cos\theta \\
\dot{u} &= -qw + \frac{D\cos\alpha - L\sin\alpha}{m} + \frac{T\cos\epsilon}{m} - g\sin\theta \\
\dot{w} &= qu - \frac{D\sin\alpha + L\cos\alpha}{m} - \frac{T\sin\epsilon}{m} + g\cos\theta \\
\dot{\theta} &= q \\
\dot{q} &= -\frac{T\sin\epsilon x_{cg}}{I_{yy}} \\
\dot{m} &= \frac{-T}{g_0 I_{sp}}
\end{aligned} \tag{2}$$

In Equation 2, $x$ and $z$ are the positions relative to the landing location, and $u$ and $w$ are the velocity components of the vehicle in the body-fixed coordinate systems $x$ and $z$ axes, respectively. The $\theta$ and $q$ are the pitch angle and the pitch angle rate, respectively. $T$ represents the thrust magnitude of the booster and $\epsilon$ is the thrust vector angle. The aerodynamic drag and lift forces are the $D$ and $L$, respectively. $I_{yy}$ and $x_{cg}$ denote the moment of inertia of the booster and the booster's center of gravity position measured from the bottom of the booster(moment arm). $g0$ represents the standard gravity and $I_{sp}$ is the thrust specific impulse. In this study, the aerodynamic moment contributions were considered negligible compared to the moment produced by the thrust vector control.

During the booster landing phase, constraints exist on the thrust magnitude and vector angle and are given in Equations 3 and 4:

$$T_{min} \leq T \leq T_{max} \tag{3}$$

$$\epsilon_{min} \leq \epsilon \leq \epsilon_{max} \tag{4}$$

In Equations 3 and 4, $T_{min}$ and $T_{max}$ correspond to the minimum and maximum applicable thrust magnitude, respectively. In addition, owing to the system requirements, $\epsilon_{min}$ and $\epsilon_{max}$ represent the minimum and maximum allowable thrust vector angles, respectively. The aerodynamic forces acting on the booster were calculated as follows:

$$D = \frac{1}{2}\rho|V|^2 S_{ref} C_D \tag{5}$$

$$L = \frac{1}{2}\rho|V|^2 S_{ref} C_L \tag{6}$$

In Equations 5 and 6, The $D$ and $T$ are the aerodynamic drag and thrust forces, respectively, $\rho$ is the air density; $|V|$ represents the velocity magnitude of the landing vehicle. $S_{ref}$ represents the reference area for the vehicle. $C_D$ and $C_L$ denote aerodynamic drag and lift coefficients, respectively. This study considers that the aerodynamic coefficients are only functions of the angle of attack: $C_D \approx C_D(\alpha)$ and $C_L \approx C_L(\alpha)$. The drag coefficient is assumed to be a quadratic function of the angle of attack,$C_D = b_0+b_1\alpha+b_2\alpha^2$, whereas the lift coefficient is linearly related to the angle of attack, $C_L = a_0 + a_1\alpha$.

In this research, in addition to the upper and lower limit constraints on the vehicle, rate limits are added to the system states and control variables to be consistent with real-world applications (given in Equations 7, 8 and 9).

$$q_{min} \leq q \leq q_{max} \tag{7}$$

$$\dot{\epsilon}_{min} \leq \dot{\epsilon} \leq \dot{\epsilon}_{max} \tag{8}$$

$$\dot{T}_{min} \leq \dot{T} \leq \dot{T}_{max} \tag{9}$$

**TABLE 1. Initial and terminal time boundary conditions.**

| | Min. Init. | Max. Init. | Terminal |
|---|---|---|---|
| **Down Range,** $x[m]$ | -2400 | -2000 | 0 |
| **Altitude,** $z[m]$ | -4000 | -4000 | 0 |
| **Velocity,** $u[\frac{m}{s}]$ | -220 | -260 | 0 |
| **Velocity,** $w[\frac{m}{s}]$ | 0 | 0 | 0 |
| **Pitch Angle,** $\theta[°]$ | 140 | 150 | 90 |
| **Pitch Rate,** $q[\frac{\circ}{s}]$ | 0 | 0 | 0 |
| **Thrust,** $T[\frac{N}{s}]$ | free | free | free |
| **TVA,** $\epsilon[°]$ | free | free | free |
| **Mass,** $m$, $[kg]$ | 55 000 | 55 000 | free |

in Equations 7, 8 and 9, $q$ is the pitch angle rate, $\epsilon$ is the thrust vector angle, and $\dot{T}_{max}$, $\dot{T}_{min}$, $q_{min}$, $q_{max}$, $\dot{\epsilon}_{min}$, $\dot{\epsilon}_{max}$ show the minimum-maximum thrust magnitude rate, minimum-maximum pitch rates, and minimum-maximum thrust-vector-angle rates respectively. The boundary conditions, including the initial and final conditions of the system states and control parameters, are detailed in Table 1.

### 3) PERFORMANCE INDEX

The formulation of an optimal control problem requires a performance index. In this study, the performance index was chosen to ensure the booster achieves soft touch-down conditions using the minimum possible fuel. This is ensured by maximizing the final mass, though efforts are made to minimize the negative impact of the final mass. A mathematical definition of the performance index is shown in Equation 10.

$$J = -m(t_f) \tag{10}$$

### 4) OPTIMAL CONTROL PROBLEM FORMULATION

Given the system dynamics, state and control constraints, path constraints, and boundary conditions, the free final time optimal control problem associated with the minimum fuel vertical rocket landing is outlined in Equation 11:

$$\begin{aligned} \min_{x,u,t_f} \quad & J = m_{fuel} \\ s.t. \quad & Equations(2) - (9) \\ & t_f = Free \end{aligned} \tag{11}$$

### 5) DATASET GENERATION STRATEGY

After defining an optimal control problem (OCP) as in Equation 11, we solved it using the RSOPT solver [36], which employs an adaptive pseudo-spectral collocation method to transcribe the continuous-time optimal control problem into discrete optimization problems and then solve the associated problem using a nonlinear programming problem solver.

In order to create a trajectory database corresponding to the dataset that will be used in the training loops of the machine learning methods, we split each initial condition interval listed in Table 1 into eight evenly-spaced samples. Therefore, 512 different initial condition combinations are attained. Then, each of the corresponding separate optimal control problems are solved and stored. Since the OCP solver RSOPT is an adaptive solver with a mesh refinement algorithm, an individual optimal trajectory rollout consists of a variable number of discrete time points (In general, it is around 100-150 discrete time points). At each discrete time point, we have an optimal value for states and control inputs described in Equation 2. If we think that the states are the features (6 states) and control inputs (2 control inputs) are the labels, in each optimal trajectory rollout, we have $m$ rows of feature vector, $m \times n_x$, and $m$ rows of label vector, $m \times n_u$, where $n_x$,$n_u$ are the number of states and number of control inputs respectively, $m$ is the number of resultant discrete time points. Now, if we consider the whole training trajectory database including 512 distinct trajectories, we have $(\sum_{i=1}^{512} m_i)$ rows of feature vector, $(\sum_{i=1}^{512} m_i) \times n_x$, and $(\sum_{i=1}^{512} m_i)$ rows of label vector, $(\sum_{i=1}^{512} m_i) \times n_u$, where $m_i$ is the resultant number of discrete time points in trajectory $i$. A representative optimal trajectory rollout is provided for the initial condition combination corresponding to the first grid point in the initial condition grid, as presented in Equations 12 and 13, as shown at the bottom of the next page. Here, $\mathcal{X}$ and $\mathcal{Y}$ denote the feature vector rollout and label vector rollout, respectively. $t_0, t_1, t_2 \ldots t_m$ indicates the discrete time points resulting from the optimal control problem solution. Throughout the remainder of this study, when reference is made to the initial condition being sampled from the trajectory database, it implies that the associated rollout will be utilized as data for training purposes.

Similarly, in addition to the training trajectories, we sampled 64 distinct initial condition combinations from the intervals specified in Table 1. Subsequently, fuel-optimal trajectories corresponding to these conditions were generated and stored in the trajectory database for validation. Figure 3 illustrates the state and control trajectories optimizing fuel consumption, which are stored in the trajectory database.

## B. SOLUTION OF THE PROBLEM WITH CLASSICAL MACHINE LEARNING ALGORITHMS

This section briefly reviews the use and employment of classical machine learning approaches in our problem. In classical machine learning approaches, a model is trained to predict a target variable using data attributes. These attributes are referred to as features. That is, the role of the trained model is to map a given feature vector of $\mathcal{X}$ to a target vector of $Y$.

In this study, as described above, the problem is to find a parametric guidance policy that maps the system states to control inputs $\pi : X \rightarrow U$ to enable the booster to land a pre-determined desired location in a fuel-optimal manner. Hence, we investigated whether using classical machine learning approaches to determine a parametric policy would be an appropriate choice, as given in Equation 14.

$$U^*(t) = \begin{bmatrix} |T|(t) \\ \epsilon(t) \end{bmatrix} = \pi(X(t), p) \tag{14}$$

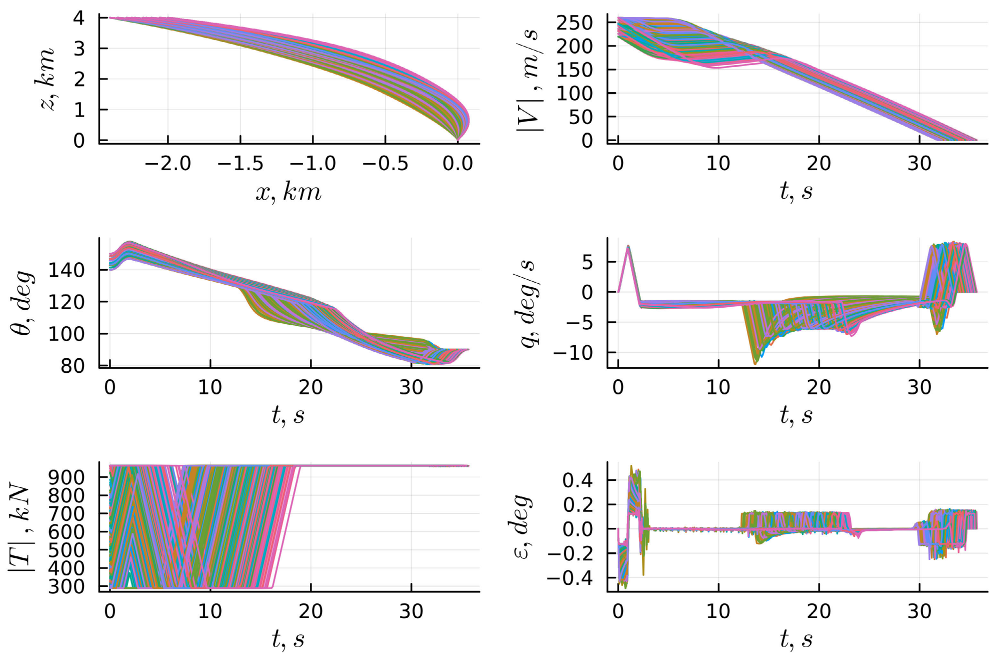

**FIGURE 3.** **Fuel optimal state trajectories generated from OCP solver.**

In Equation 14, $U^*(t)$ is the system's time-varying optimal control input vector that tried to be solved, $T$ is thrust magnitude, $\epsilon$ is thrust vector angle, $\pi$ is the parametric guidance policy, $X(t) \in \mathbb{R}^6$ is the system's state vector at the corresponding time step,($[x(t), z(t), u(t), w(t), q(t), \theta(t)]$), and $p$ is the parameters of the policy. The flow diagram in Figure 4 summarizes the utilized pipeline for the classical machine learning methods used in this study. We used data from the preliminary trajectory database obtained by solving the problem using an OCP solver to train a policy model, including various trajectories for training. Each consists of a discrete number of optimal state-action pairs along with time. These pairs can be regarded as data in the format of $(\mathcal{X} - \mathcal{Y})$ pairs. Hence, we contemplated that developing a fully data-driven model using optimal state-action pairs with classical machine learning methods could result in a policy that attempts to map the system states to actions in an optimal manner. In the first step of the training process, the data stored in the database were preprocessed by normalization and standardization to eliminate unwanted gradient behaviors and outliers in the data. For this purpose, we used min-max scaling to scale the raw data as follows:

$$\mathbb{X}^{sc} = \frac{\mathbb{X}^{raw} - \mathbb{X}^{raw}_{min}}{\mathbb{X}^{raw}_{max} - \mathbb{X}^{raw}_{min}} \tag{15}$$

$$\mathcal{X}^{(1)}(t)^* = \begin{array}{c} \\ t_0 \\ t_1 \\ t_2 \\ \vdots \\ t_m \end{array} \begin{array}{cccccc} x^*, m & z^*, m & u^*, \frac{m}{s} & w^*, \frac{m}{s} & \theta^*, ^\circ & q^*, \frac{^\circ}{s} \\ \left[\begin{matrix} -2400 \\ -2394.8 \\ -2382.9 \\ \vdots \\ 0 \end{matrix}\right. & \begin{matrix} -4000 \\ -3995.7 \\ -3985.7 \\ \vdots \\ 0 \end{matrix} & \begin{matrix} -220 \\ -219.9 \\ -219.9 \\ \vdots \\ 0 \end{matrix} & \begin{matrix} 0 \\ -0.22 \\ -0.79 \\ \vdots \\ 0 \end{matrix} & \begin{matrix} 140 \\ 140 \\ 140.01 \\ \vdots \\ 90 \end{matrix} & \left.\begin{matrix} 0 \\ 0.029 \\ 0.423 \\ \vdots \\ 0 \end{matrix}\right] \end{array} \tag{12}$$

$$\mathcal{Y}^{(1)}(t)^* = \begin{array}{c} \\ t_0 \\ t_1 \\ t_2 \\ \vdots \\ t_m \end{array} \begin{array}{cc} |T|^*, N & \epsilon^*, ^\circ \\ \left[\begin{matrix} 288883.86 \\ 288877.38 \\ 288876.82 \\ \vdots \\ 961586.12 \end{matrix}\right. & \left.\begin{matrix} 0 \\ -0.12 \\ -0.40 \\ \vdots \\ 0 \end{matrix}\right] \end{array} \tag{13}$$

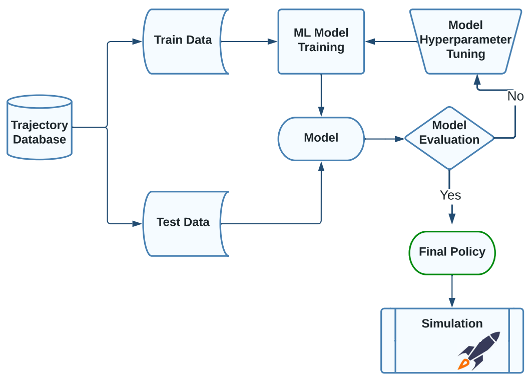

**FIGURE 4.** Policy development flow diagram for data-driven machine learning methods.

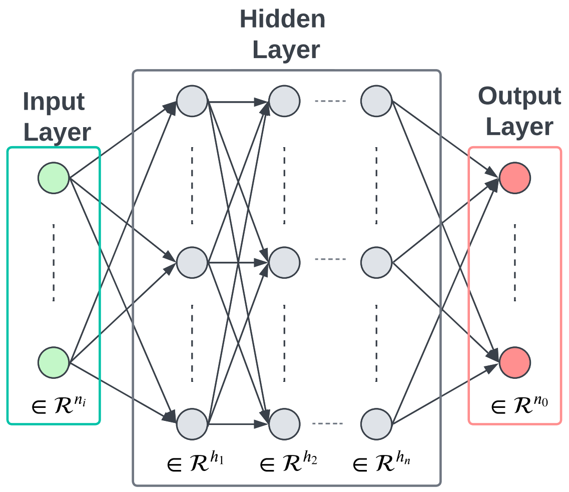

**FIGURE 5.** General deep neural network architecture.

where $\mathbb{X}^{sc}$, $\mathbb{X}^{raw}$ represent the scaled and raw data vectors, including features and labels, respectively, and $\mathbb{X}^{raw}_{min}$, $\mathbb{X}^{raw}_{max}$ denote the minimum and maximum values that exist in the raw data for both the feature and label vectors. The data were then split into test and training data using the data-twinning algorithm developed to prevent data distribution shifts [38]. Subsequently, classical machine learning methods were employed for training. Before the last step, the trained policy was evaluated using the unseen test data to measure the generalization performance of the algorithms. For all the algorithms in this study, we implemented extensive hyper-parameter search and optimization methods to enhance the policy mapping performance further [39]. The final policy is implemented in a simulation environment to investigate its performance and whether the mission requirements were satisfied.

The preliminaries of the approaches and models used in this study are described briefly in the following subsections.

## C. MULTILAYER PERCEPTRON FEED FORWARD NEURAL NETWORKS

Deep neural networks can be utilized as universal approximators for any nonlinear continuous function by using a sufficient number of neurons [40], [41]. Hence, we employed a DNN model to describe the nonlinear mapping between the state and actions of the guidance policy.

Figure 5 shows the general layout of a DNN, which consists of an input layer with an arbitrary number of features, hidden layers stacked together, and an output layer. The input-output mapping is achieved with sequential mathematical operations followed by nonlinear activation functions using a finite number of free parameters called weights and biases. The optimal combinations of these parameters were obtained using gradient-based optimization methods.

Considering $x_l$ as an input vector for a specific layer $l$, the mathematical calculation can be expressed as follows:

$$f_l = \sigma_l(\omega.x_l + b_l) \tag{16}$$

In Equation 16, $\omega_l \in \mathbb{R}^{N_l \times N_{l+1}}$ are the weight matrices, and the $N_l, N_{l+1}$ account for the number of neurons used in layers $l$ and $l+1$ respectively. The $b_l \in \mathbb{R}^{N_{l+1}}$ denotes bias vector. Finally, the $\sigma_l$ corresponds to the nonlinear activation function used in the layer. Different types of activation functions can be used in DNN models to provide nonlinearity. The most common choices are *Linear*, *Tanh*, *Relu*, *Gelu*, and *Sigmoid*. After defining the formulations for a specific layer, the final prediction of the DNN model can be constructed, as given in Equation 17.

$$\hat{y}(x) = (f_l \circ f_{l-1} \cdots f_2 \circ f_1)(x) \tag{17}$$

In Equation 17, $\circ$ denotes the composition operator, $f_l$ is the $l.th$ NN layer. As mentioned previously, the learnable parameters $W_l$ and $b_l$ are trained using backpropagation algorithms [42]. However, to implement the backpropagation algorithm, it is necessary to define a performance index. For instance, the first thing that comes to mind, as well as the common choice for regression-related tasks, is to utilize the Mean Squared Error (MSE) as a loss function. Using the definition of a loss function, the networks are trained to determine the best combinations of weights and biases. It is possible to use different variants of optimization algorithms during backpropagation, such as stochastic gradient-descent, Momentum, ADAM, and NADAM [43], according to the nature of the problem.

## D. ADAPTIVE ACTIVATION FUNCTIONS FOR NEURAL NETWORKS

The nonlinear activation function used in the layers of the DNN model transforms the input data in a nonlinear manner, enabling it to learn more complex tasks. Saturated, unsaturated, and adaptive activation functions are the three categories of activation functions used in neural network architectures [44]. The idea of adaptive activation functions (AAF) was first proposed by [45], and the parametric *ReLU* activation function, which consists of three piecewise linear functions with four learnable parameters, is described. The authors in [33] formalized the concept of adaptive activation functions and used them to accelerate the convergence of physics-informed neural networks by modifying the loss landscape of the neural network, particularly in the early training phase.

The principal idea of adaptive activation functions is to make them trainable to capture more nonlinear behavior. Therefore, they facilitate the learning of complex tasks and accelerate the convergence speed. When an activation function $\sigma(x)$ is considered, we can include a trainable parameter $\alpha$ as $\sigma(\alpha x)$, which provides additional degrees of freedom and adjusts the slope of the function. It has been claimed that adding a non-trainable scaling factor $n$ into the equation as $\sigma(n\alpha x)$ also contributes to the convergence speed [33]. These activation functions can be utilized in two distinct ways: layer-wise or neuron-wise. The former implements the same AAF throughout the layer so that only one trainable parameter exists. The latter enables the training of AAFs independently across the neurons of the layer, which results in an equal number of trainable parameters to the number of neurons used in the layer. Consequently, examples of the *tanh* and *ReLU* adaptive activation functions are shown in Equation 18.

$$\begin{aligned} Adaptive\ Tanh &:= \frac{e^{n\alpha x} - e^{-n\alpha x}}{e^{n\alpha x} + e^{-n\alpha x}} \\ Adaptive\ ReLU &:= max(0, n\alpha x) \end{aligned} \tag{18}$$

In addition to these AAFs, another has been proposed as a new class of AF called *rowdy* [34]. The *rowdy* activation function allows NNs to exploit various properties of distinct AAFs and combine them into a single function by summing them, $\sigma(x) = \sum_{k=1}^{N} \sigma_k(\alpha x)$. The authors suggested using a baseline classic activation function and including extra trigonometric activation functions with trainable parameters as additional terms [34]. The formulation is given by Equation 19:

$$\sigma(x) = \sigma_{base}(x) + \sum_{k=1}^{N} \beta_k \sin(\beta_k (k-1)x) \tag{19}$$

In Equation 19, $\sigma_{base}$ represents the baseline activation function. The $\beta_k$ is the trainable parameter for the trigonometric AAF. For illustration purposes, one can observe the behaviors of the AAFs and rowdy activation function in Figure 6. Illustrations are given for the case where $\sigma_{base}()$ is the adaptive *tanh* function with various trainable parameters $\alpha$. The parameters used for the *rowdy* activation function are $K = 2$, $\beta_1 = \beta_2 = 0.1$, $w_1 = 10$, and $w_2 = 20$. As shown in Figure 6, changing the trainable parameters of the adaptive *tanh* activation function changes the slope of the function. However, the *rowdy* function provides flexible behavior with oscillatory movements in the saturation regions of the *tanh* function. We selected the *tanh* activation function as a base activation function for our work. The main reason for this selection is that saturated activation functions do not cause

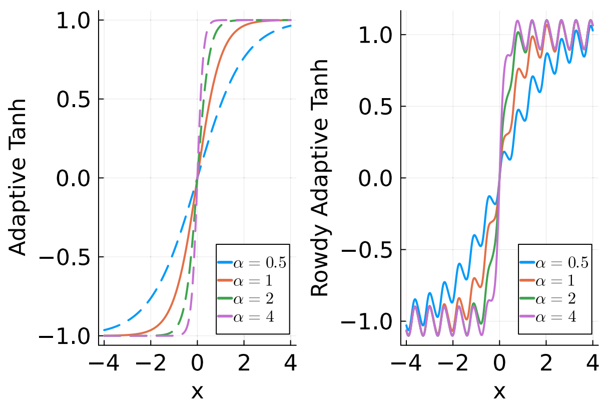

**FIGURE 6.** **Adaptive *Tanh* and *Rowdy* adaptive *Tanh* activation functions.**

an increase in gradients. The gradient is close to zero when activation functions like *sigmoid* or *tanh* saturate. Based on the mathematical properties of the *tanh* activation function, given in Equation 18, it is less likely to suffer from the exploding gradient problem because of its bounded output.

## E. QUADRATIC RESIDUAL NEURAL NETWORKS

The Quadratic Residual Network (QResNet) architecture was presented in [28], which aims to enhance the model's expressive power, enabling it to capture high-frequency responses using fewer parameters than conventional ANNs. At each network layer, QResNet delivers quadratic nonlinearity prior to the application of activation functions. The formulation of a quadratic neuron is given by Equation 20:

$$\phi_l = \sigma_l(\omega_1^l.x_l \circ \omega_2^l.x_l + \omega_1^l.x_l + b^l) \tag{20}$$

In Equation 20, $\phi_l$ is the output of layer $l$, $x_l \in \mathbb{R}_l^N$ is the input vector for layer $l$, $\omega_1^l \in \mathbb{R}^{N_l \times N_{l+1}}$ and $\omega_2^l \in \mathbb{R}^{N_l \times N_{l+1}}$ are the weight matrices at layer numbers $l$ and $l+1$. The $N_l, N_{l+1}$ account for the number of neurons used in layers $l$ and $l+1$ respectively, and $\circ$ denotes the Hadamard product. The term $(\omega_1^l.x_l \circ \omega_2^l.x_l)$ is the quadratic residual term and the $b^l \in \mathbb{R}^{N_{l+1}}$ is the bias vector. Finally, the $\sigma_l$ corresponds to the nonlinear activation function used in the layer.

$$\hat{\Phi}(x) = (\phi^{(l)} \circ \phi^{(l-1)} \cdots \phi^{(2)} \circ \phi^{(1)})(x) \tag{21}$$

In Equation 21, $\hat{\Phi}$ is the output of the QResNet, $\circ$ denotes the composition operator, $\phi^{(l)}$ is the output of the *lth* QResNet layer, $x$ is the input vector for the first layer. As mentioned previously, the learnable parameters $\omega_1^l$, $\omega_2^l$, and $b^l$ can be trained with the backpropagation algorithm using gradient-based optimization methods.

## F. QUADRATIC RESIDUAL NEURAL NETWORKS WITH ADAPTIVE ADAPTIVE ACTIVATION FUNCTIONS

We developed two new types of QResNet called Adaptive Quadratic Residual Neural Networks (AQResNet) to speed up the training process, improve convergence, increase prediction accuracy by increasing NN capacity, learn higher

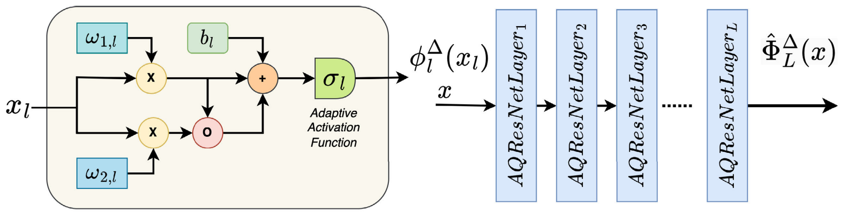

**FIGURE 7.** **Adaptive quadratic residual neural networks (a) adaptive QResNet layer block diagram (b) quadratic residual neural networks with adaptive activation function block diagram.**

frequencies in regression problems, and reduce the size of the network's parameters. We accomplished this by presenting adaptive activation functions [32], [33] to QResNet so that the NN could benefit from additional trainable parameters and provide it with a greater degree of freedom. We also suggested another type of QResNet called Rowdy Adaptive Quadratic Residual Neural Networks (RAQResNet), which uses special trigonometric functions [34] in the activation functions with particular arithmetic operations to smooth the training loss surface, provide parameter efficiency and prevent falling into local optima during network training. In Figure 7, the general framework of the adaptive QResNet layer and multilayer adaptive quadratic neural network architecture is shown.

In this framework, we can define feed-forward architecture by using a QResNet layer and an adaptive activation function as given in Equations 22 and 23:

$$\phi_l^\Delta = \sigma_l^\Delta(\omega_1^l.x_l \circ \omega_2^l.x_l + \omega_1^l.x_l + b^l) \tag{22}$$

In Equation 22, $\phi_l^\Delta$ is the output of layer $l$, $x_l \in \mathbb{R}_l^N$ is the input vector for layer $l$, $\omega_1^l \in \mathbb{R}^{N_l \times N_{l+1}}$ and $\omega_2^l \in \mathbb{R}^{N_l \times N_{l+1}}$ are the weight matrices at layers $l$ and $l+1$. The $N_l, N_{l+1}$ account for the number of neurons used in layers $l$ and $l+1$ respectively, $\circ$ denotes the Hadamard product, and the $b^l \in \mathbb{R}^{N_{l+1}}$ is the bias vector. Finally, the $\sigma_l^\Delta$ corresponds to the nonlinear *adaptive* activation function used in the layer $l$.

$$\hat{\Phi}_L^\Delta(x) = (\phi_l^\Delta \circ \phi_{l-1}^\Delta \cdots \phi_2^\Delta \circ \phi_1^\Delta)(x) \tag{23}$$

In Equation 23, $\hat{\Phi}_L^\Delta(x)$ is the output of the adaptive QResNet, $\circ$ denotes the composition operator, $\phi_l^\Delta$ is the *lth adaptive* QResNet layer, $x$ is the input vector for the first layer.

For the first type of AQResNet, the *tanh* adaptive activation function is given by Equation 24:

$$\sigma_l^\Delta(x) = \frac{e^{n_l\alpha_l x} - e^{-n_l\alpha_l x}}{e^{n_l\alpha_l x} + e^{-n_l\alpha_l x}} \tag{24}$$

In Equation 24, $\sigma_l^\Delta$ is the adaptive activation function at layer $l$, $\alpha_l$ is a trainable parameter that provides an additional degree of freedom and adjusts the slope of the function at layer $l$. It is claimed that adding a non-trainable scaling factor $n$ into the equation as $(n\alpha)$ also contributes to the convergence speed [33]. Therefore, we retained this term as a separate hyper-parameter.

In the original work [34] of the rowdy activation function, the authors proposed that the selection of $\sigma_{base}$ may be one of the *Tanh*, *ReLU*, *Elu*, *Sigmoid*, *Gelu*, and *Swish* activation functions and then added extra trigonometric terms (as given in Equation 19) to form a rowdy activation function. In this definition, only the $\beta_l$ parameters are trainable and can be learned from the data during NN training. However, in our study, we have made some changes to define a fully adaptive rowdy activation function. First, we replaced the base activation part with Equation 24 so that the $\sigma_{base,l}^\Delta$ part of the equation can be adaptive and learned from the data as well (it can be replaced by any adaptive version of the classical activation functions). Second, we introduced an additional training parameter for the trigonometric part of Equation 19 with $\Omega_l$. Consequently, the frequency of the trigonometric part of the rowdy activation function is also adaptive and learnable from the data during training. Thus, all parts of the $\sigma_l^\Delta(x)$ become adaptable and trainable.

More specifically, as a second type of AAF, we propose RAQResNet and its mathematical expression is given in Equations 25 and 26.

$$\sigma_l^\Delta(x) = \sigma_{base,l}^\Delta(x) + \sum_{k=1}^{N} \psi_{l,k}^\Delta(x_l) \tag{25}$$

$$\psi_{l,k}^\Delta(x) = \beta_l \sin((k-1)\Omega_l x_l) \tag{26}$$

In Equations 25 and 26, $\sigma_{base,l}^\Delta$ represents the baseline activation function at layer $l$, and $\psi_{k,l}^\Delta$ is the trigonometric part of the AAF in layer $l$. The $\beta_l$ and $\Omega_l$ are the trainable parameters for the trigonometric part of the AAF in layer $l$, and $k = 1, \cdots, K; K \in \mathbb{N}$ is the frequency adjusting/involving parameter for the trigonometric function terms.

In summary, we can train a quadratic neural network using Equations 22 and 23, by defining the activation function $\sigma_l^\Delta$ as in Equation 24 for AQResNet and by defining the activation function $\sigma_l^\Delta$ in Equation 25 for RAQResNet. Training can be performed using the standard backpropagation algorithm with first and second-order optimization methods such as ADAM, RMSProb, and L-BFGS.. etc. The final form of RAQResNet for the *adaptive* $-$ *tanh* activation function is given by Equation 27:

$$\sigma_l^\Delta(x) = \frac{e^{n_l\alpha_l x} - e^{-n_l\alpha_l x}}{e^{n_l\alpha_l x} + e^{-n_l\alpha_l x}} + \sum_{k=1}^{N} \beta_l \sin((k-1)\Omega_l x) \tag{27}$$

As stated earlier, the learnable parameters $\omega_1^l$, $\omega_2^l$, and $b^l$ can be trained with the backpropagation algorithm using gradient-based optimization methods. In addition, in our framework, new trainable parameters for adaptive activation must be included in the training. The resulting optimization problem entails minimizing the loss function by optimizing both the newly introduced parameters and the NN weights and biases. The training parameters for AQResNet and RAQResNet are given in Equations 28 and 29, respectively:

$$\Upsilon_{AQResNet} = \left\{\omega_1^l, \omega_2^l, b^l, \alpha^l\right\}_{l=1}^{L-1} \tag{28}$$

$$\Upsilon_{ARQResNet} = \left\{\omega_1^l, \omega_2^l, b^l, \alpha^l, \beta^l, \Omega^l\right\}_{l=1}^{L-1} \tag{29}$$

In Equations 28 and 29, $\omega_1^l$ and $\omega_2^l$ are the weight matrices of layer $l$, $b^l$ is the bias vector of layer $l$, $\alpha^l$, $\beta^l$, and $\Omega^l$ are the trainable parameters of layer $l$ for adaptive activation functions.

Now, our optimization problem can be defined for additional parameters to train an AQResNet and can be given by Equation 30. Then, we can update the $\alpha^l$ parameter using the loss function and its gradients by using the gradient descent step with Equation 31.

$$\alpha^*_{AQResNet_l} = \underset{\alpha_l \in \mathbb{R}^+}{\arg\min}(\mathcal{L}(\alpha)) \tag{30}$$

$$\alpha^l_{m+1} = \alpha^l_m - \eta_l \nabla_{\alpha^l} \mathcal{L}_m(\Upsilon_{AQResNet}) \tag{31}$$

In Equations 30 and, 31, $\alpha^*_{AQResNet_l}$ is the optimal value of $\alpha$ at layer $l$, $\mathcal{L}$ is the loss function, $\mathcal{L}_m$ is the loss value at the iteration step $m$, $\eta_l$ is the learning rate, and $m$ is the number of iterations. $\alpha^l_m$, is the trainable parameter value at the iteration step $m$ in the layer $l$. $\nabla_{\alpha^l}$ is the gradient operator for the Loss function with respect to $\alpha$ parameter at layer $l$, and $l = 1, \cdots, L$; $L \in \mathbb{N}$ is the layer number.

Similarly, we can define an optimization problem for the additional parameters to train RAQResNet, which is given by Equation 32. Again, we can update the $\alpha^l$, $\beta^l$, and $\Omega^l$ parameters using the loss function and its gradients using the gradient descent step with Equation 33.

$$\begin{aligned}
\alpha^*_{RAQResNet_l} &= \underset{\alpha_l \in \mathbb{R}^+}{\arg\min}(\mathcal{L}(\omega_1, \omega_2, b, \alpha, \beta, \Omega)) \\
\beta^*_{RAQResNet_l} &= \underset{\beta_l \in \mathbb{R}^+}{\arg\min}(\mathcal{L}(\omega_1, \omega_2, b, \alpha, \beta, \Omega)) \\
\Omega^*_{RAQResNet_l} &= \underset{\Omega_l \in \mathbb{R}^+}{\arg\min}(\mathcal{L}(\omega_1, \omega_2, b, \alpha, \beta, \Omega))
\end{aligned} \tag{32}$$

$$\begin{aligned}
\alpha^l_{m+1} &= \alpha^l_m - \eta_l \nabla_{\alpha^l} \mathcal{L}_m(\Upsilon_{RAQResNet}) \\
\beta^l_{m+1} &= \beta^l_m - \eta_l \nabla_{\beta^l} \mathcal{L}_m(\Upsilon_{RAQResNet}) \\
\Omega^l_{m+1} &= \Omega^l_m - \eta_l \nabla_{\Omega^l} \mathcal{L}_m(\Upsilon_{RAQResNet})
\end{aligned} \tag{33}$$

In Equations 32 and 33, $\alpha^*_{RAQResNet_l}$, $\beta^*_{RAQResNet_l}$, $\Omega^*_{RAQResNet_l}$ are the optimal values of $\alpha$, $\beta$ and $\Omega$ trainable parameters at layer $l$, $\mathcal{L}$ is the loss function, $\mathcal{L}_m$ is the loss value at iteration step $m$, $\eta_l$ is the learning rate, and $m$ is the iteration number, $\alpha^l_m$, $\beta^l_m$, and $\Omega^l_m$ are the trainable parameter values at the iteration step $m$ in the layer $l$. $\nabla_{\alpha^l}$, $\nabla_{\beta^l}$, $\nabla_{\Omega^l}$ are the gradient operators for the loss function with respect to $\alpha$, $\beta$ and $\Omega$ parameters at layer $l$, and $l = 1, \cdots, L$; $L \in \mathbb{N}$ is the layer number.

## G. MODELING OF THE PROBLEM WITH SCIENTIFIC MACHINE LEARNING

The scientific machine learning concept was defined in [25] to tackle the generalization problem of fully data-driven models in physical applications by incorporating physics phenomena governing the data into the training processes. To determine an appropriate policy, the booster landing problem is re-formulated as a scientific model discovery problem similar to that described in [25]. To achieve this goal, a guidance policy can be treated as a learnable unknown dynamical model that functions as the system states. Considering this, the mechanistic motion model given in Equation 2 can be decomposed into known-unknown dynamics and represented as universal ordinary differential equations (UODE). The unknown part is modeled with a universal approximator which is chosen as an artificial neural network parameterized with $\theta_{NN}$, which includes the network weights and biases.

$$\pi(X(t), p) = \begin{bmatrix} |T|(t) \\ \epsilon(t) \end{bmatrix} \approx \mathcal{NN}(X(t), \theta_{NN}) \tag{34}$$

In Equation 34, $X(t) \in \mathbb{R}^6$ denotes the current state vector at the corresponding time step,($[x(t), z(t), u(t), w(t), q(t), \theta(t)]$). Hence, the mechanistic motion model given in Equation 2 is rewritten as follows;

$$\begin{aligned}
\dot{x} &= u\cos\theta + w\sin\theta \\
\dot{z} &= -u\sin\theta + w\cos\theta \\
\dot{u} &= -qw + \frac{D\cos\alpha - L\sin\alpha}{m} - g\sin\theta \\
&\quad + \frac{\mathcal{NN}^{(1)}(X(t), \theta_{NN})\cos(\mathcal{NN}^{(2)}(x(t), \theta_{NN}))}{m} \\
\dot{w} &= qu - \frac{D\sin\alpha + L\cos\alpha}{m} + g\cos\theta \\
&\quad - \frac{\mathcal{NN}^{(1)}(X(t), \theta_{NN})\sin(\mathcal{NN}^{(2)}(X(t), \theta_{NN}))}{m} \\
\dot{\theta} &= q \\
\dot{q} &= -\frac{\mathcal{NN}^{(1)}(X(t), \theta_{NN})\sin(\mathcal{NN}^{(2)}(X(t), \theta_{NN}))x_{cg}}{I_{yy}} \\
\dot{m} &= \frac{-\mathcal{NN}^{(1)}(X(t), \theta_{NN})}{g_0 I_{sp}}
\end{aligned} \tag{35}$$

where, In $\mathcal{NN}^{(*)}()$, "*" shows the $*_{th}$ output of the neural network.

## H. PERFORMANCE INDEX AS LOSS FUNCTIONS

With this proposed UODE, learning the unknown policy corresponds to training the parameterized artificial neural network, $\mathcal{NN} : \mathbb{R}^6 \rightarrow \mathbb{R}^2$. To do so, we define a performance index that enforces the fuel optimality condition in conjunction with the path and boundary constraints.

Therefore, it is considered to have a cost function as a performance index that consists of the following three parts:

$$\mathcal{L}_{total} = \lambda_f \mathcal{L}_{fuel} + \lambda_p \mathcal{L}_{PC} + \lambda_{BC} \mathcal{L}_{BC} \tag{36}$$

In Equation 36, $\mathcal{L}_{fuel}$ is the cost that considers the distance from the optimal value to fuel consumption. $\mathcal{L}_{PC}$ deals with path constraints to prevent trajectories from evolving into infeasible directions. $\mathcal{L}_{BC}$ ensures that the boundary conditions are satisfied. $\lambda_f, \lambda_p, \lambda_{bc}$ are the corresponding cost coefficients used to scale the importance levels of the cost contributors in the optimization loop.

In this study, we considered handling fuel consumption criteria and path constraints by adding a continuous cost term that minimizes the $L_2$ norm of the distance between the system states and the corresponding optimal values obtained from the solution of the OCP. By reducing this distance, the policy optimization process is expected to converge to the fuel optimal trajectories because optimal trajectories are obtained by minimizing fuel consumption.

$$\mathcal{L}_{fuel} = \lambda_f \left|\left|X(t)_{opt} - X(t)_{NN}\right|\right| \tag{37}$$

In Equation 37, $X(t)_{opt} \in \mathbb{R}^6$ represents the optimal state vector at the corresponding time step, $([x(t), z(t), u(t), w(t), q(t), \theta(t)])$, and $X(t)_{NN} \in \mathbb{R}^6$ shows the current state vector under a neural network driven policy.

The continuous angle of the attack path constraint should be applied to avoid diverging trajectories in the policy optimization loop. We employed this condition using ReLU functions, which only add cost when the minimum-maximum limits are violated, and the final mathematical form of this approach is given in Equation 38:

$$\mathcal{L}_{PC} = \lambda_p(||ReLU(\alpha(t) - \alpha_{max}) + ReLU(-\alpha(t) - \alpha_{min})||) \tag{38}$$

Although minimizing the $L_2$ norm between system states and optimal states contains information about boundary conditions, we consider defining additional quadratic costs for enforcing the booster to satisfy the soft landing conditions with high accuracy. These conditions are given in Equation 39 as a loss function form.

$$\begin{aligned}\mathcal{L}_{BC} = \lambda_{BC_1}((x(t_f) - x_{BC})^2 + (z(t_f) - z_{BC})^2) \\ + \lambda_{BC_2}((u(t_f) - u_{BC})^2 + (w(t_f) - w_{BC})^2) \\ + \lambda_{BC_3}(\theta(t_f) - \theta_{BC})^2\end{aligned} \tag{39}$$

where $\lambda_{BC_1}, \lambda_{BC_2}, \lambda_{BC_3}$ are the position, velocity, and attitude cost coefficients, respectively, which are used to scale the importance level of each contributor term.

## I. SIMULATION AND SCIML OPTIMIZATION LOOP

The UODE implementation and the resulting policy optimization loop were performed using the Julia programming language [46]. We used highly sophisticated and advanced native implementations for ODE solvers, in addition to ad-joint methods for automatic differentiation algorithms using this programming language.

In this study, the ODE solutions were obtained using a combination of solvers Tsit5 and Rosenbrock23, which are native Julia implementations of the Tsitouras 5/4 Runge-Kutta and L-Stable Rosenbrock-W methods. The algorithm determines which method to use in specific parts of the ODE according to the stiffness of the problem. The training policy approximator neural network was performed using Julia library called DiffEqFlux.jl [25]. We developed several simulation loops to obtain gradient information with automatic differentiation for ODE systems fused with parametric models such as neural networks.

The training loop given in the SciML flowchart in Figure 8 can be briefly explained as follows: The initial conditions' training grid is given as input to the algorithm, accompanied by the associated fuel-optimal trajectory solutions from the trajectory database. The first step of the algorithm involves initializing neural network weights. After successful initialization, mini-batches are randomly formed from the initial condition grid. This implies that the mini-batches contain various trajectories corresponding to distinct initial conditions. Then, a simulation is run for each initial condition in the mini-batches, which results in the calculation of the $\mathcal{L}_{total}$ given in Equation 36 along with the policy gradient with respect to $\mathcal{L}_{total}$. After completing all the initial conditions in a mini-batch, the average loss and gradient calculations are performed. This makes conducting gradient descent over the policy network possible. Well-known ADAM optimizer [47] was selected for this study. After updating the network weights for all mini-batches, a new epoch starts. At the beginning of a new epoch, mini-batches are created randomly. The same steps are followed until convergence is achieved. Convergence can be evaluated based on the user preferences.

In this study, a fixed epoch size was used as the stopping condition. Finally, when convergence was achieved, the network policy was evaluated with unseen initial conditions in the test trajectories to determine whether any over-fitting conditions existed. Otherwise, the final policy is obtained. If not, a hyper-parameter tuning process was performed for the parametric model.

Algorithm 1 summarizes the policy optimization process with the SciML approach by using UDOE.

The constant inputs of the optimization algorithm are the training grid of the initial conditions defined in Table 1, the mini-batch size $M$, which corresponds to the number of trajectories evaluated in one step of the optimization algorithm, and the number of epochs $N$, where the optimization algorithm is maintained. The data used in the optimization algorithm were retrieved from the trajectory database obtained from the solution of the associated optimal control problem defined in Section III-A4.

The first step of the algorithm is the initialization of the neural network model, which is the guidance policy mapping the current system states to the control actions that can be observed in Equation 35. After successful initialization

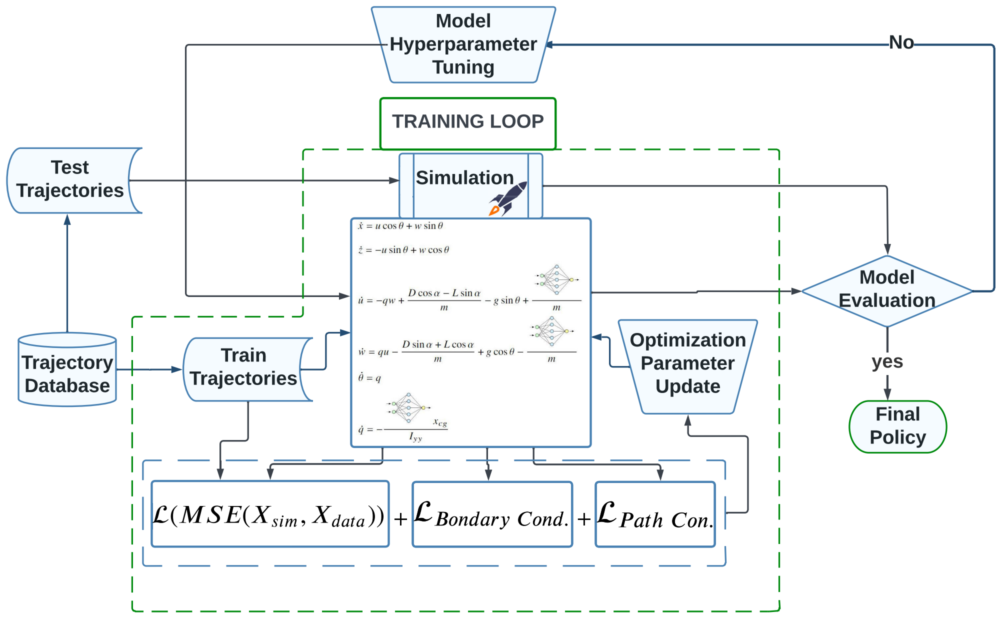
Model Hyperparameter Tuning
No
TRAINING LOOP
Test Trajectories
Simulation
Model Evaluation
Trajectory Database
Train Trajectories
Optimization Parameter Update
yes
Final Policy
$\mathcal{L}(MSE(X_{sim}, X_{data})) + \mathcal{L}_{Bondary\ Cond.} + \mathcal{L}_{Path\ Con.}$

**FIGURE 8.** Policy development flowchart for scientific machine learning methodology.

**Algorithm 1** SciML Minibatch Training Algorithm

```
1  Input: ODE Solver(ODE, IC's), Epoch Size: N, Mini-Batch Size: M
2  Data: Pre-Generated Optimal Trajectories
3  Initialization:Initilization of Network Weights, $\theta_{NN}$
4  Optimization Variable:, $\theta_{NN}$
5  for $epoch = 1, 2, \ldots, N$ do
6      Generate Minibatches
7      for $Minibatch = 1, 2, \ldots$ do
8          for Trajectories in Minibatch do
9              $x(t)_i, u(t)_i \leftarrow$ Run the $simulation(IC_i, \theta_{NN})$
10             Compute Loss, $\mathcal{L}_i \leftarrow x(t)_i$
11             Compute Gradient, $\left(\frac{d\mathcal{L}_i}{d\theta_{NN}}\right)_i \leftarrow \mathcal{L}_i, x(t)_i, u(t)_i, \theta_{NN}$
12         end for
13         Compute Mean Loss, $\mathcal{L}_{mean} = \frac{1}{M}\Sigma_{i=1}^{M}\mathcal{L}_i$
14         Compute Mean Gradient :
15         $\left(\frac{d\mathcal{L}}{d\theta_{NN}}\right)_{mean} = \frac{1}{M}\Sigma_{i=1}^{M}\left(\frac{d\mathcal{L}_i}{d\theta_{NN}}\right)_i$
16         Gradient Descent with Adam Optimizer :
17             $\theta_{updated} \leftarrow \mathcal{L}_{mean}, \left(\frac{d\mathcal{L}}{d\theta_{NN}}\right)_{mean}$
18         Update Network Weights, $\theta_{NN} \leftarrow \theta_{updated}$
19     end for
20 end for
```

of the neural network weights $\theta_{NN}$ corresponding to the optimization variable, based on the user preference input value of the mini-batch size $M$, the mini-batches are formed randomly from the initial condition grid. In other words, mini-batches incorporate various trajectories corresponding to distinct initial conditions.

Subsequently, a numerical simulation was conducted for each initial condition in the mini-batches, corresponding to the time integration of the universal ordinary differential equations given in Equation 35. At the end of each simulation, the algorithm computes the value of the loss function $\mathcal{L}_{total}$ as defined in Equation 36. The policy gradient with respect to the total loss $(\frac{d\mathcal{L}_{total}}{d\theta_{NN}})$ was then computed using automatic differentiation facilitated by the Julia library DiffEqFlux.jl [25]. The average loss and gradient calculations were performed after completing all initial conditions in a mini-batch. In addition, the algorithm employed gradient descent with an ADAM optimizer [47] to update the policy network weights. A new epoch begins after updating the network weights for all mini-batches. At the start of each epoch, mini-batches were randomly created. These steps are repeated until the algorithm reaches the user-specified epoch value, $N$.

Finally, once convergence was achieved, the network performance was evaluated using the unseen initial conditions from the test trajectories stored in the trajectory database to assess the presence of overfitting. If no overfitting conditions are detected, the final optimized policy is obtained. Consequently, the algorithm outputs the optimized neural network weights $\theta^*_{NN}$. Therefore, the final form of the guidance policy, with optimized neural network weights $\theta^*_{NN}$, is:

$$\begin{bmatrix} |T|^*(t) \\ \epsilon^*(t) \end{bmatrix} = \mathcal{NN}(X(t), \theta^*_{NN}) \tag{40}$$

In Equation 40, $X(t) \in \mathbb{R}^6$ represents the current state vector at the corresponding time step, $[x(t), z(t), u(t), w(t), q(t), \theta(t)]$.

## IV. SIMULATIONS AND RESULTS

This section briefly explains the simulations conducted for the policy performance evaluations and model comparisons. After optimizing each policy and implementing it into the simulations, evaluations and comparisons were carried out based on the soft landing requirements and fuel consumption metrics. Our simulations consisted of several case studies, including nominal training trajectories, validation trajectories, out-of-domain trajectories, and Monte Carlo analyses. The performances of the policies in all the cases are first investigated on nominal trajectories, and comparisons are conducted; then, out-of-domain performance and uncertainty analysis are performed to assess the robustness of the policies. To the best of our knowledge, this is the most extensive analysis in this domain that uses the SciML approach. We used *policy* or *policies* terms to identify trained neural network models for SciML, Classical ML, and the NN models with adaptive activation functions for the remaining sections. Our experiments were performed on a Macbook Pro (2017) equipped with a 2.3 GHz dual-core Intel Core i5 processor and 8 GB of 2133 MHz LPDDR3 RAM. We briefly explained all the steps for the simulations in the following sections:

### A. PARAMETERS FOR SIMULATIONS

Considering the mechanistic motion model given in Equation 2, we obtained and used similar system parameters, rocket aerodynamics, and the parameters utilized in flight dynamics as in [36] for the experiments. This study considers that the aerodynamic coefficients are only functions of the angle of attack: $C_D \approx C_D(\alpha)$ and $C_L \approx C_L(\alpha)$. The drag coefficient is assumed to be a quadratic function of the angle of attack,$C_D = b_0 + b_1\alpha + b_2\alpha^2$, whereas the lift coefficient is linearly related to the angle of attack, $C_L = a_0 + a_1\alpha$. In this work, on top of upper and lower limit constraints on the vehicle, it is also considered to add rate limits to system states and control variables to be consistent with real-world applications (given in Equations 7, 8 and 9). The nominal, minimum, and maximum values stated in the constraint equations were employed in the simulations/experiments, as listed in Table 2.

We used the training and testing trajectories stored in the database. As previously described in detail in Section III-A5, it consists of optimal trajectories originating from the initial conditions grid. We formed these initial condition combinations by splitting each interval listed in Table 1 into eight evenly-spaced samples. Thus, 512 distinct fuel optimal trajectories were acquired by solving the resultant optimal control problems using the RSOPT solver [36] and stored in the trajectory database. Moreover, 64 different initial condition combinations were sampled randomly from the intervals listed in Table 1, and the corresponding fuel optimal trajectories were generated by solving optimal control problems with RSOPT and stored in the trajectory database for testing purposes.

**TABLE 2.** **Constant parameters and bounds for variables in simulation.**

| Parameter | Value |
|---|---|
| **Moment of Inertia,** $I_{yy}, kgm^2$ | 75000 |
| **Moment Arm,** $x_{cg}, m$ | 5 |
| **Reference Area,** $S, m^2$ | 12.54 |
| **Thrust Specific Impulse,** $I_{sp}, s$ | 443 |
| **Max Thrust Magnitude,** $T_{max}$**,** $kN$ | 1375.6 |
| **Thrust Magnitude,** $T$ | $T \in [20\%, 70\%].T_{max}$ |
| **Thrust Magnitude Rate,** $\dot{T}$ | $\dot{T} \in [-\frac{T_{max}}{4}, \frac{T_{max}}{4}]$ |
| **Pitch Rate,** $q$ | $q \in [-10\frac{\circ}{s}, 10\frac{\circ}{s}]$ |
| **Angle of Attack,** $\alpha$ | $\alpha \in [-25^\circ, 25^\circ]$ |
| **Thrust Vector Angle,** $\epsilon$ | $\epsilon \in [-3^\circ, 3^\circ]$ |
| **Thrust Vector Angle Rate,** $\dot{\epsilon}$ | $\dot{\epsilon} \in [-6\frac{\circ}{s}, 6\frac{\circ}{s}]$ |
| **Aerodynamic Coeffs.,** $[a_0, a_1]$ | $[-0.05, 0.007]$ |
| **Aerodynamic Coeffs.,** $[b_0, b_1, b_2]$ | $[0.25, 2 * 10^{-4}, 6 * 10^{-4}]$ |

### B. MODEL ARCHITECTURES FOR NEURAL NETWORKS

In this study, we implemented three different NN architectures for a performance comparison between the vanilla neural network and our proposed NN-based novel architectures. To create a fair benchmark, we fixed the layer size and total number of parameters used; thus, the neurons in the layers were selected accordingly. All the NN models and their architectural details are listed in Table 3. In Table 3, $n_i$ represents the number of neurons used in layer *i*, and the LA and NA abbreviations account for layer-wise adaptive and neuron-wise adaptive, respectively. From Table 3, it can be seen that we have scaling and re-scaling operations before the first hidden layer and after the last hidden layer to avoid unwanted gradient behaviors through the layers during gradient-based optimization, such as vanishing-exploding gradients. Scaling operations were performed using the min-max scaling method, as given in Equation 15. Using the Tanh function before the last layer makes it possible to bind the output variables between $[-1, 1]$, and then the re-scaling operation is performed such that the interval $[-1, 1]$ is mapped to $[y_{min}, y_{max}]$, which enables the policy not to produce control outputs that violate the minimum and maximum bounds of the system. The mapping formula is given by Equation 41.

$$y_m = y_{l_3}\frac{y_{max} - y_{min}}{2} + \frac{y_{max} + y_{min}}{2} \tag{41}$$

where $y_m$ is the mapped(re-scaled) output, and $y_{l_3}$ represents the variable vector resulting from layer-3 for all architectures. It is essential to note that although we used layer-wise adaptive activation functions through the inner layers, we used neuron-wise adaptive activation functions in the last layer of our proposed architectures. The nature of the output variables $|T|$ and $\epsilon$ and their dynamical behaviors are completely different, which can be observed from the optimal control problem solutions given in Figure 3. Therefore, using neuron-wise adaptive activation functions is considered to contribute to achieving higher accuracy levels by adapting independently themselves to particular dynamics.

**TABLE 3.** Neural network architectures for SciML.

| | Vanilla NN | | | AQResNet | | | RAQResNet | | |
|---|---|---|---|---|---|---|---|---|---|
| **Layer** | **Type** | **# units** | **Activation** | **Type** | **# units** | **Activation** | **Type** | **# units** | **Activation** |
| Input | Scaling | 6 | - | Scaling | 6 | - | Scaling | 6 | - |
| Layer1 | Dense | $6 * n_1$ | Tanh | QRes | $2 * 6 * n_1$ | LA-Tanh | Qres | $2 * 6 * n_1$ | Rowdy |
| Layer2 | Dense | $n_1 * n_2$ | Tanh | QRes | $2 * n_1 * n_2$ | LA-Tanh | Qres | $2 * n_1 * n_2$ | Rowdy |
| Layer3 | Dense | $n_2 * 2$ | Tanh | Dense | $n_2 * 2$ | NA-Tanh | Dense | $n_2 * 2$ | NA-Tanh |
| Output | Re-Scaling | 2 | - | Re-Scaling | 2 | - | Re-Scaling | 2 | - |

### C. HYPER-PARAMETERS

For the scientific machine learning optimization loop, hyper-parameter selections, including loss coefficients $\lambda$s, mini-batch-size $M$, were performed based on a grid search. After executing a grid search to find the optimal hyper-parameters, the loss coefficients discovered to have the best performance were $\lambda_f = 0.1$, $\lambda_p = 1$, $\lambda_{BC_1} = 5$, $\lambda_{BC_2} = 1$ and $\lambda_{BC_3} = 1$. Learning rate annealing was used in this study to boost training performance and reduce training time. As mentioned previously, a fixed number of epochs was employed to stop the optimization loop, which was $epoch_{max} = 3000$. Hence, the initial learning rate $\eta_0$ was set to 0.001. Subsequently, the learning rate schedule was applied such that the learning rate was halved every 500 epochs, which resulted in $\eta_{final} = 3.125 * 10^{-5}$.

The optimization pseudo code given in Algorithm 1 includes the hyper-parameter of the mini-batch size, $M$. Our simulations showed that the optimized policy (trained NN models) performs much better when utilizing stochastic gradient descent (SGD) over trajectories for this particular problem, such as having more minor training errors and better generalization abilities. This means that in a single epoch consisting of 512 training trajectories, for each trajectory, gradient descent optimization was applied as presented in Algorithm 1. Hence, we used a mini-batch size of 1, $M = 1$, and all the results were obtained accordingly.

### D. MODEL TRAINING RESULTS

During the model training process, the initial conditions were sampled from the database, and a tolerance value of $10^{-6}$ was used for both absolute and relative errors in the simulations. The objective of the training was to minimize the loss function defined in Equation 36. As mentioned before, all models were trained with a fixed number of epochs in the SciML optimization loop, which was $epoch_{max} = 3000$. Accordingly, the initial learning rate $\eta_0$ was set to 0.001. Subsequently, the learning rate schedule was applied such that the learning rate was halved every 500 epochs, which resulted in $\eta_{final} = 3.125 * 10^{-5}$. We found that the execution time of our policy optimization loop was approximately 12-13 hours. The average training and validation error histories over (loss values) epochs, including 512 distinct training trajectories and 64 different validation trajectories, are shown in Figure 9 for the policy architectures given in Table 3. In Figure 9, in addition to the proposed architectures, we presented two NN models with feed-forward architectures named Vanilla NN-1 and Vanilla NN-2 results. The Vanilla NN-1 model had the same number of total parameters as our proposed models; in contrast, the Vanilla NN-2 had twice the total number of parameters. From Figure 9, the learned policies with our proposed architectures AQResNet and RAQResNet in which only 16 neurons were used in the hidden layers (equivalent to 772 and 780 total parameters respectively) quickly converged to smaller training and validation error values compared with the vanilla-NN architectures. The RAQResNet policy achieved the smallest training error of $\approx$ 4.0. The AQResNet policy reached an error value of $\approx$ 10.0 while the Vanilla NN-2 policy stayed around 250.0, and the Vanilla NN-1 model could not achieve a training loss of less than 1000. These results show that our proposed NN architectures can capture nonlinear, complex system behaviors with much better accuracy (roughly 50-300 times better) and have a better learning capacity than Vanilla-NNs. It can also be inferred that RAQResNet has more promising learning/training performance (almost 2.5 times better) than AQResNet.

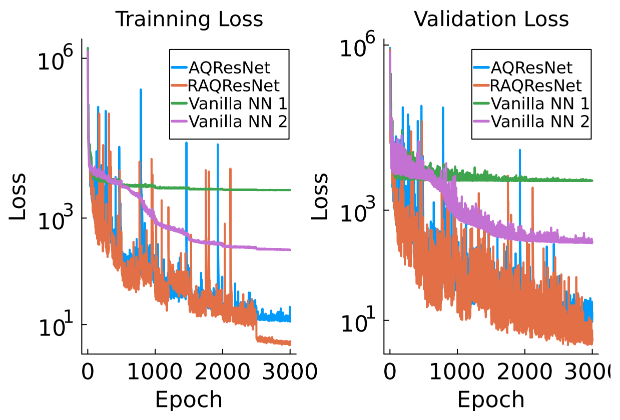

**FIGURE 9.** Training and validation loss histories for SciML models.

The final adaptive activation function states in the layers of the AQResNet and RAQResNet models after training are shown in Figure 10. In Figure 10, $Layer(i)$ denotes the resulting optimized adaptive activation function in $i^{th}$ layer. Layer $3^{(1)}$ and Layer $3^{(2)}$ represent the trained neuron-wise adaptive tanh function in Layer3 for the first control variable, $|T|$ and the second control variable, $\epsilon$ respectively. Considering the trained neuron-wise adaptive activation functions, we contemplated that our assumption regarding the use of neuron-wise adaptive activation functions in the last layer to better capture the different natures of outputs was

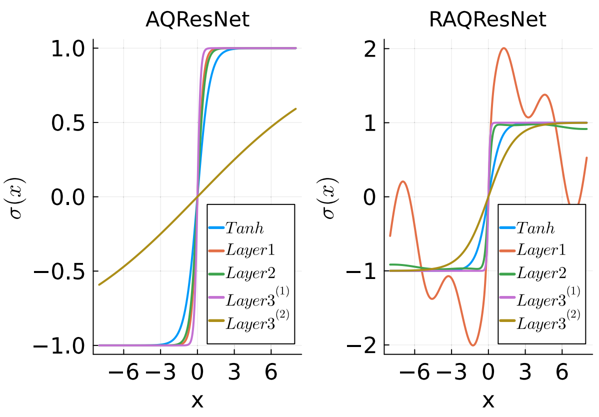

**FIGURE 10.** Final activation functions.

a wise choice. The figure shows that the trained neuron-wise adaptive activation function in the last layer of the AQResNet architecture for thrust vector angle $\epsilon$ exhibits almost linear behavior within the depicted range. In contrast, the resulting optimized neuron-wise adaptive activation function for thrust magnitude $|T|$ behaves like a hard tanh function, which is quite different from linear behavior. Therefore, it can be concluded that using a distinct adaptive activation function for neurons in the last layer of the architectures worked as intended. On the other hand, it can be observed from the figure that RAQResNets' trained adaptive activation function at Layer1 is highly oscillatory because of the trigonometric functions utilized in the adaptive rowdy activation function.

### *E. CLASSIC MACHINE LEARNING AND SCIML METHODS PERFORMANCE COMPARISON RESULTS*

We evaluated the performance of the learned policies (NN models) by implementing them in simulations. We compared their efficiency and accuracy by investigating how well they achieved soft and precise landing conditions, along with the fuel optimality criteria. For this part, we also trained an additional gradient boosting algorithm (XGBoost) to compare the NNs' prediction capability with another type of machine learning algorithm. Table 4 lists the error statistics for the simulated training and validation trajectories in the database for the optimized policies(trained models) attained by the scientific and classical machine learning approaches.

Table 4 lists the statistics of the error values of the touch-down conditions (TD), landing location error $\Delta x$, landing velocity magnitude error $\Delta|V|$, and landing attitude angle error $\Delta\theta$, along with the fuel consumption statistics $\Delta m_{fuel}$. Moreover, it contains information on the worst-case scenarios. In this table, $\mu, \sigma$, and $\gamma$ represent the mean, standard deviation, and worst-case values, respectively. When the table is investigated, it can be easily observed that all the policies derived from the classical machine learning approach with different methods failed to fulfill the soft-precise landing conditions in the training trajectories because they have significant error residuals for every condition except the fuel usage value. However, there is no point in using less fuel when mission requirements can not be satisfied. Even though these policies were trained with optimal state-action pairs and achieved good training metrics during training, they failed to perform well when implemented in simulations, even in training trajectories. This is because the data obtained from the OCP have a discrete number of points, while the simulation works in continuous time. Consequently, models face a considerable amount of unseen data or unseen data distribution regions, which are prone to produce action values with errors. Even though the generated action values have a minor error at a specific time step, they exhibit incremental growth when integration continues, resulting in large errors at the final time. However, the policies trained by the scientific machine learning approach performed particularly well, except for the vanilla neural network architectures, which is an expected situation because of the training error history given in Figure 9. This implies that our proposed architectures have much more expressive power than the vanilla architecture, even if more trainable parameters are introduced. Although the vanilla neural network policy succeeded at a certain level of performance, more is required. However, the policies with our proposed architectures AQResNet and RAQResNet accomplished a remarkable triumph. The policies trained with the scientific machine learning approach differ from those trained with the classical machine learning method because the policies learn not only the data but also the physics law governing the data. Therefore, even if some error occurs at specific time steps, the policies learn how to reduce the error while proceeding with the time steps because of the learning underlying physics information.

When the error statistics are considered for our proposed policies(NN models), the soft-precision landing conditions are satisfied with minor error residuals for both the training and validation trajectories that are not seen during training. Hence, it can be said that both our proposed policies have satisfactory generalization performance on unseen data and trajectories.

When the performance of both policies (trained models) is considered, Table 4 concludes that the RAQResNet policy achieved better performance for both simulated trajectory sets. For instance, trajectories driven by the AQResNet policy have the mean landing position error of 0.36 and 0.42 in training and validation trajectories. On the other hand, the mean landing position error values of 0.18 and 0.2 for the training and validation trajectories are driven by the RAQResNet policy. Moreover, the RAQResNet policy is superior to the AQResNet policy in terms of touch-down velocity magnitude errors. The mean velocity magnitude errors for the RAQResNet policy are equal to 0.35 and 0.37 in training and validation trajectories, respectively, while the mean errors for the AQResNet policy are 0.94 and 0.99, respectively. Although the performance of the RAQResNet policy is superior, that of AQResNet is also acceptable because the error residuals lie in the acceptable regions.

**TABLE 4.** Model performance comparison.

| METHOD | MODEL | METRIC | TRAINING SET $\mu$ | TRAINING SET $\sigma$ | TRAINING SET $\gamma$ | VALIDATION SET $\mu$ | VALIDATION SET $\sigma$ | VALIDATION SET $\gamma$ |
|---|---|---|---|---|---|---|---|---|
| SCI-ML | Vanilla NN | $\Delta x, m$ | 1.39 | 1.31 | 8.3 | 1.65 | 1.55 | 5.7 |
| | | $\Delta\lvert V\rvert, \frac{m}{s}$ | 4.77 | 2.68 | 19.46 | 4.86 | 2.67 | 14.79 |
| | | $\Delta\theta, ^\circ$ | 2.84 | 1.98 | 7.83 | 3.17 | 2.18 | 7.81 |
| | | $\Delta m_{fuel}, kg$ | 6070.35 | 153.57 | 6515.39 | 6070.45 | 172.13 | 6478.77 |
| | AQResNet | $\Delta x, m$ | 0.361 | 0.33 | 1.5 | 0.43 | 0.39 | 1.43 |
| | | $\Delta\lvert V\rvert, \frac{m}{s}$ | 0.94 | 0.19 | 1.62 | 0.94 | 0.2 | 1.43 |
| | | $\Delta\theta, ^\circ$ | 0.23 | 0.19 | 1.68 | 0.23 | 0.22 | 1.3 |
| | | $\Delta m_{fuel}, kg$ | 6087.77 | 167.16 | 6561.50 | 6088.52 | 184.52 | 6514.23 |
| | RAQResNet | $\Delta x, m$ | 0.18 | 0.06 | 0.5 | 0.2 | 0.08 | 0.52 |
| | | $\Delta\lvert V\rvert, \frac{m}{s}$ | 0.35 | 0.21 | 1.11 | 0.37 | 0.21 | 1.02 |
| | | $\Delta\theta, ^\circ$ | 0.1 | 0.08 | 0.48 | 0.11 | 0.10 | 0.45 |
| | | $\Delta m_{fuel}, kg$ | 6094.92 | 167.28 | 6563.03 | 6095.34 | 185.21 | 6526.96 |
| CLASSICAL-ML | Linear Reg. | $\Delta x, m$ | 127.57 | 183.73 | 871.43 | - | - | - |
| | | $\Delta V, \frac{m}{s}$ | 148.37 | 74.16 | 195.62 | - | - | - |
| | | $\Delta\theta, ^\circ$ | 32.48 | 33.57 | 89.42 | - | - | - |
| | | $\Delta m_{fuel}, kg$ | 3267.45 | 1390.63 | 5995.72 | - | - | - |
| | Vanilla NN | $\Delta x, m$ | 203.87 | 179.73 | 761.38 | - | - | - |
| | | $\Delta V, \frac{m}{s}$ | 132.45 | 69.07 | 198.09 | - | - | - |
| | | $\Delta\theta, ^\circ$ | 42.44 | 25.15 | 84.72 | - | - | - |
| | | $\Delta m_{fuel}, kg$ | 3462.86 | 1390.63 | 5908.64 | - | - | - |
| | XGBOOST | $\Delta x, m$ | 98.53 | 96.80 | 663.1 | - | - | - |
| | | $\Delta V, \frac{m}{s}$ | 68.06 | 29.51 | 185.16 | - | - | - |
| | | $\Delta\theta, ^\circ$ | 51.31 | 22.12 | 89.24 | - | - | - |
| | | $\Delta m_{fuel}, kg$ | 4995.55 | 530.95 | 5912.77 | - | - | - |

Regarding the worst-case conditions, from the table, the maximum position error for the AQResNet policy is seen at the training trajectories, which is equal to 1.5. For the RAQResNet policy, the maximum position error is 0.52. Both the values were considered acceptable. The worst velocity magnitude errors are 1.62 and 1.11 for the AQResNet and RAQResNet, respectively. We regard these values as reasonable because an additional velocity controller can tackle them in the last few seconds of the landing. Finally, the worst touch-down angle errors were 1.68° and 0.4°, which are again considered acceptable because the help of the landing legs can compensate for these attitude errors.

To evaluate the generalization performance of the proposed policies further, we conducted an out-of-domain simulation. However, the policies trained according to the initial position range of $[-2400, -2000]$ given in Table 1 and the trained policies were tested with the initial positions that lie entirely outside of this interval. To do so, the left and right-hand sides of the given range were extended by delta amount $\delta$, and the initial conditions were sampled only from this extended domain, $x_0 \in [-\delta - 2400, -2400] \cup [-2000, -2000 + \delta]$. Figure 11 illustrates the in-domain and out-of-domain trajectories driven by the AQResNet and RAQResNet policies.

Table 5 summarizes the error statistics for the touch-down conditions (TD) and fuel consumption statistics for the out-of-domain simulations controlled by the AQResNet and RAQResNet policies.

From Table 5, it can be observed that both trained policies achieved good performance, even in the out-of-domain trajectories. However, it should be noted that the AQResNet policy could only tolerate the $\pm 50m$ domain extension. After these values, our experiments showed that failing trajectories started to appear. In contrast, the RAQResNet policy showed the ability to tolerate $\pm 150, m$ domain extension with great accuracy. Hence, it can be inferred that the policies trained by the scientific machine learning approach have a good generalization capability even in out-of-distribution cases. In the following section, we discuss the fuel optimality performance by comparing the fuel consumption statistics with optimal control solutions.

**TABLE 5.** Out of domain statistics.

| Model | Metric | $\mu$ | $\sigma$ | $\gamma$ |
|---|---|---|---|---|
| AQResNet $\delta = \pm 50, m$ | $\Delta x, m$ | 0.34 | 0.23 | 1.64 |
| | $\Delta\lvert V\rvert, \frac{m}{s}$ | 1.36 | 1.11 | 4.6 |
| | $\Delta\theta, ^\circ$ | 0.66 | 0.53 | 2.79 |
| | $\Delta m_{fuel}, kg$ | 6084.17 | 172.05 | 6562.53 |
| RAQResNet $\delta = \pm 150, m$ | $\Delta x, m$ | 0.16 | 0.09 | 0.93 |
| | $\Delta\lvert V\rvert, \frac{m}{s}$ | 0.65 | 1.12 | 4.53 |
| | $\Delta\theta, ^\circ$ | 0.22 | 0.33 | 2.46 |
| | $\Delta m_{fuel}, kg$ | 6102.22 | 188.37 | 6552.33 |

## F. THE OPTIMAL CONTROL PERFORMANCE COMPARISON FOR THE MODELS WITH ADAPTIVE ACTIVATION FUNCTIONS

The results presented in the previous section proved that the scientific machine learning approach is superior to the classical machine approach for generating a guidance policy for the booster landing problem. Hence, in this section and the next section, the results will only be shared for the policies obtained with the AQResNet and RAQResNet architectures trained using scientific machine learning.

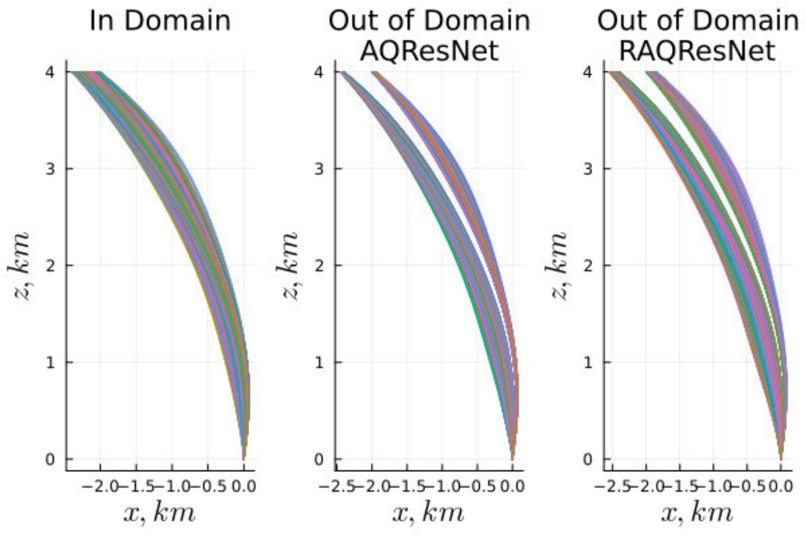

**FIGURE 11.** Trajectories driven by (R)AQResNet neural network models.

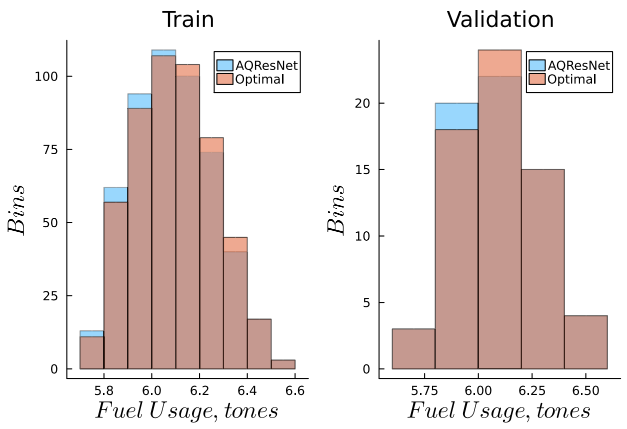

**FIGURE 12.** Fuel consumption histograms for AQResNet.

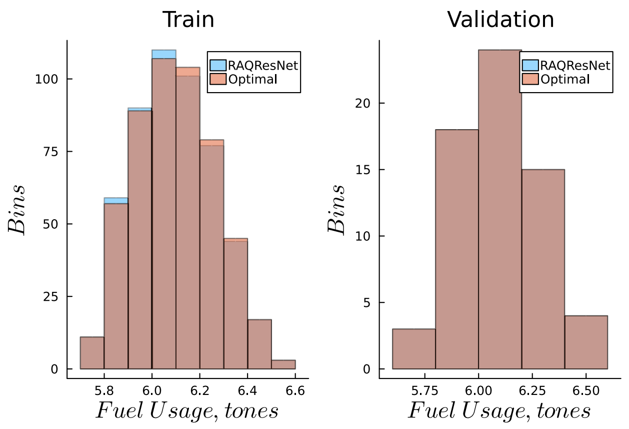

**FIGURE 13.** Fuel consumption histograms for RAQResNet.

The fuel consumption histograms resulting from the training and validation trajectory simulations driven by the AQResNet policy are shown in Figure 12. This figure also contains information on the fuel consumption values of the corresponding trajectories contained in the trajectory database obtained from the optimal control problem solution.

Similarly, Figure 13 shows the fuel consumption histograms of the training and validation trajectories under the control of the RAQResNet policy and the corresponding optimal fuel usage histograms.

Table 6 summarizes the fuel consumption statistics shared in Figures 12-13 resulting from simulations implementing the policies with our proposed architectures, along with the optimal fuel consumption values attained from the OCP solutions stored in the trajectory database.

From Figures 12-13 and Table 6, it can be concluded that the policies have near-optimal fuel consumption performance. Indeed, it appears that the mean fuel consumption values of the trajectories implemented in our proposed policies are lower than the optimal values. However, the reason for this is that our proposed policies bring the booster to the landing location with minor velocity magnitude residuals. Consequently, this causes slightly less fuel consumption than the optimal ones because the fuel required for decelerating to zero velocity is not used, unlike the optimal control solution in which the landing velocities are exactly equal to zero.

Three trajectories were sampled from the validation set in the trajectory database. Figures 14 and 15 illustrate the evolution of the system states and control actions produced by AQResNet and RAQResNet, respectively, for the same trajectories.

Figures 14 and 15 show that the optimized policies with the AQResNet and RAQResNet architectures learned how to control the booster in a fuel-optimal manner because the system states evolved over time about the same as the optimal solutions. However, the optimal actions taken by

**TABLE 6.** Fuel consumption comparison with optimum values.

| | | Policy Optimality Comparison | | | | | |
|---|---|---|---|---|---|---|---|
| | | **AQResNet Policy** | | **RAQResNet Policy** | | **RSOPT (Nominal)** | |
| | | Train | Validation | Train | Validation | Train | Validation |
| **Δ** m [kg] | $\mu$ | 6087.77 | 6088.52 | 6094.92 | 6095.34 | 6098.31 | 6098.68 |
| | $\sigma$ | 167.16 | 184.52 | 167.28 | 185.21 | 168.23 | 186.47 |
| | *Worst* | 6561.50 | 6514.23 | 6563.03 | 6526.96 | 6562.24 | 6528.96 |

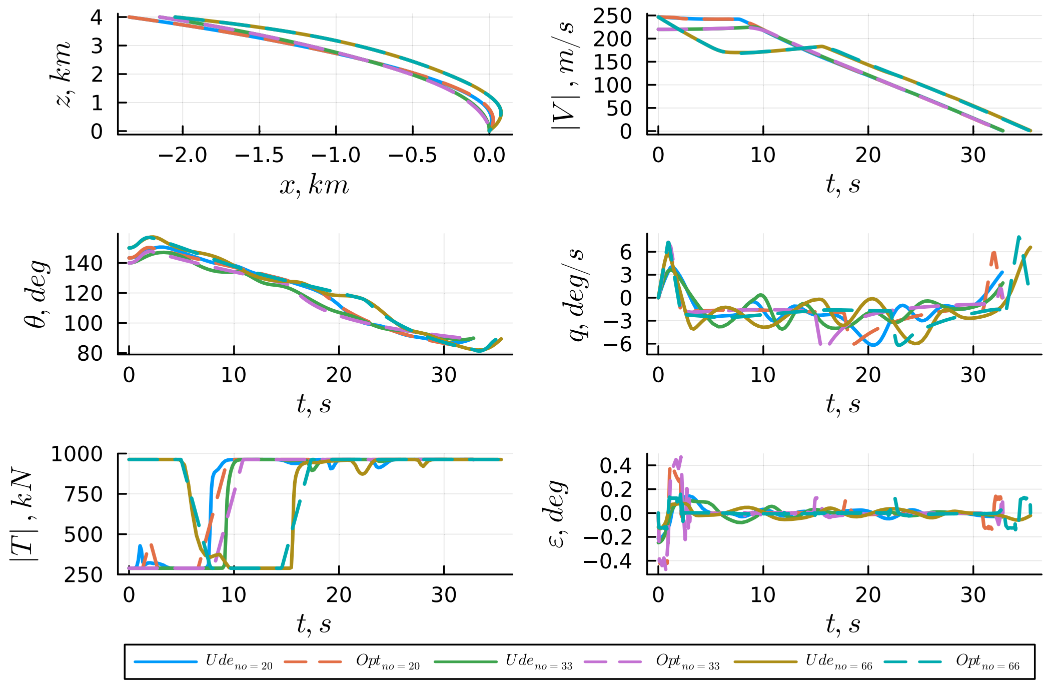

**FIGURE 14.** Trained network state trajectories for validation set - AQResNet.

the optimized policies differ slightly from each other and from the optimal solutions. Although the general thrust profiles appear similar, the optimized policies have more bang-bang type control, unlike the optimal solutions. On the other hand, when we consider the thrust vector angle profiles and the actions taken by the AQResNet policy are smoother than those taken by RAQResNet and the optimal solution. This situation also resulted in slightly different pitch rate evaluations. However, the overall performance of the optimized policies can be regarded as near optimal. Even if the actions taken by the policies vary from the optimal solutions at some time steps, they re-arranged the actions in the future to fulfill the mission while being on the fuel-optimal trajectory.

Considering the real-time computation ability of our proposed models, since we could not find a chance to implement our proposed algorithm on an actual flight computer, we embedded our trained networks on a single-board computer named Raspberry Pi 4 using the Armadillo library [48] to assess the real-time implementation applicability. Through rigorous testing, we measured the inference times of our models, revealing that they could execute each guidance step within microseconds. The results are presented in Table 7. These findings prove that the proposed NNs do not impose an additional computational burden on the guidance hardware system. Hence, this emphasizes the feasibility and practicality of real-time implementation.

**TABLE 7.** Inference times of guidance neural networks.

| AQResNet | RAQResNet |
|---|---|
| $53.08\mu s \pm 3.05\mu s$ | $57.13\mu s \pm 4.23\mu s$ |

## G. RESULTS OF THE MONTE CARLO ANALYSIS

We conducted Monte Carlo simulations to assess the performance and evaluate the robustness of our proposed guidance approaches against uncertainties and disturbances, including NN models with AQResNet and RAQResNet architectures trained with scientific machine learning. We implemented different scenarios with the 3 degrees of freedom flight model for the uncertainty and disturbance analysis. We performed 1000 Monte Carlo simulations for all cases presented below.

**Case 1:** The minimum and maximum values of the initial conditions listed in Table 1 are assumed to be uniformly

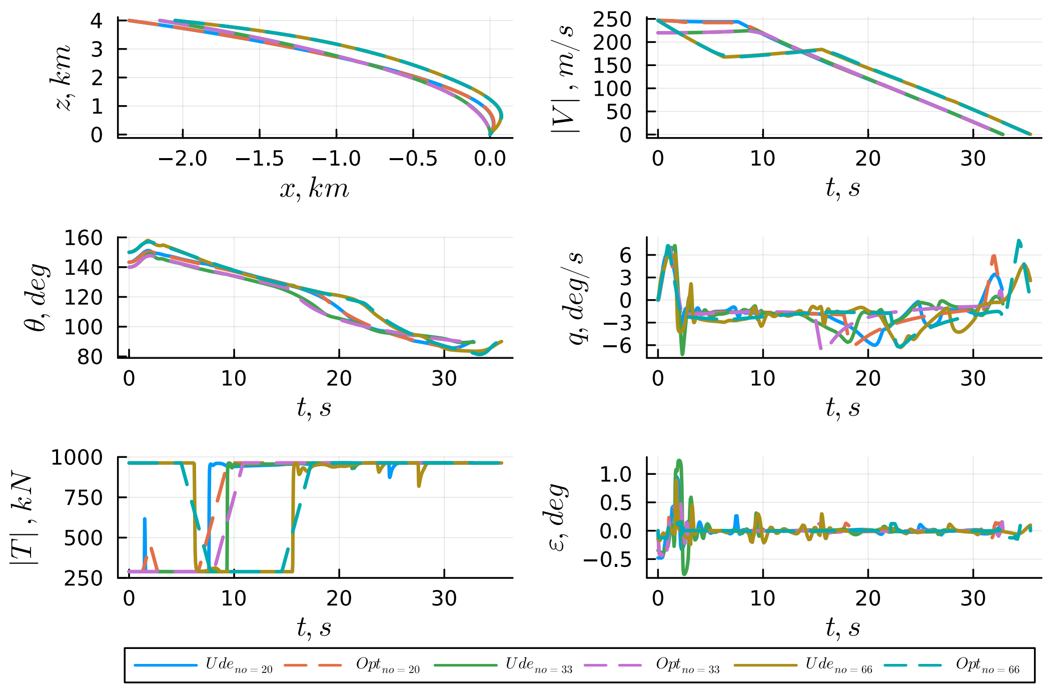

**FIGURE 15.** Trained network state trajectories for validation set - RAQResNet.

distributed between these values. The initial conditions were sampled randomly from this distribution.

$$X_0 \sim \mathcal{U}(x_{0_{min}}, x_{0_{max}}) \tag{42}$$

where $X_0$ is the initial condition vector comprising the state variables and $\mathcal{U}(min, max)$ denotes the uniform distribution over the minimum bound $min$ and maximum bound $max$.

**Case 2:** In addition to the Case-1 scenario, multiplicative $\pm 10\%$ uncertainty was given to the drag coefficient, whereas the lift coefficient was taken as nominal.

$$C_D = C_{D_{nom}} * \mathcal{U}(0.9, 1.1) \tag{43}$$

where $C_{D_{nom}}$ shows the nominal value of the drag coefficient.

**Case 3:** Unlike the Case-2 scenario, this time $\pm 10\%$ uncertainty is given to the lift coefficient while the drag coefficient is taken as nominal.

$$C_L = C_{L_{nom}} * \mathcal{U}(0.9, 1.1) \tag{44}$$

where $C_{L_{nom}}$ shows the nominal value of the lift coefficient.

**Case 4:** On top of the Case-1 scenario, $\pm 10\%$ uncertainty is given to both the drag and lift coefficients.

**Case 5:** Together with the uncertainties existing in Case-4, measurement noise was added to the state variables as follows:

$$S = S + |S| \times \mathcal{N}(0, 0.01) \tag{45}$$

where $S$ denotes the state variables and $\mathcal{N}(\mu, \sigma)$ represents the normal distribution with mean $\mu$ and standard deviation $\sigma$. It should be noted that noise magnitudes are restricted to the upper and lower limits owing to the assumption of using filters to eliminate noise before the guidance and control loop. The noise on the position states is limited to $\pm 1m$. Noise on velocity components is limited $\pm 0.5\frac{m}{s}$, noise on pitch angle $\theta$ measurement is bounded by $\pm 0.5^\circ$ and the noise on the pitch angle rate $q$ is limited to the $\pm 0.5\frac{\circ}{s}$. $I_{sp_{nom}}$ represents the nominal value of the thrust-specific impulse.

**Case 6:** In addition to the Case-5 scenario, $\pm 5\%$ thrust specific impulse $I_{sp}$ uncertainty was added to account for the unmodeled thrust dynamics.

$$I_{sp} = I_{sp_{nom}} * \mathcal{U}(0.95, 1.05) \tag{46}$$

The Monte Carlo analysis results for the optimized policies AQResNet and RAQResNet, which were trained only for the nominal conditions, are tabulated in Table 8 for each case described above. This table summarizes the touch-down error and fuel consumption statistics for the worst-case scenarios.

In Table 8, $\mu$, $\sigma$ and $\gamma$ denote the mean, standard deviation, and worst case values, respectively. From Table 8, both the proposed policies achieved outstanding performance in Case-1 to Case-4. The uncertainties introduced in the simulation did not affect the touch-down metrics or the fuel consumption values, except for the worst-case values for the touch-down velocities. However, when we evaluate the worst touch-down velocity magnitudes, it can be seen that they are lower than 5 $\frac{m}{s}$, which is indeed compensated magnitudes with additional controllers that can be employed simultaneously with our policies. On the other hand, when measurement noises are added to the system variables, the touch-down performance metrics start to degrade but not

**TABLE 8.** Monte carlo study results for proposed approach.

| Uncertainty | Metric | AQResNet $\mu$ | AQResNet $\sigma$ | AQResNet $\gamma$ | RAQResNet $\mu$ | RAQResNet $\sigma$ | RAQResNet $\gamma$ |
|---|---|---|---|---|---|---|---|
| **Case-1** | $\Delta x, m$ | 0.27 | 0.26 | 1.17 | 0.18 | 0.05 | 0.42 |
| | $\Delta\lvert V\rvert, \frac{m}{s}$ | 1.32 | 1.04 | 3.7 | 0.12 | 0.15 | 1.86 |
| | $\Delta\theta, ^\circ$ | 0.57 | 0.37 | 1.86 | 0.13 | 0.15 | 1.86 |
| | $\Delta m_{fuel}, kg$ | 6080.98 | 138.28 | 6517.24 | 6103.43 | 152.98 | 6540.99 |
| **Case-2** | $\Delta x, m$ | 0.29 | 0.27 | 1.3 | 0.19 | 0.06 | 0.58 |
| | $\Delta\lvert V\rvert, \frac{m}{s}$ | 1.39 | 1.09 | 3.94 | 0.24 | 0.49 | 4.53 |
| | $\Delta\theta, ^\circ$ | 0.58 | 0.41 | 2.04 | 0.18 | 0.29 | 3.62 |
| | $\Delta m_{fuel}, kg$ | 6079.21 | 146.77 | 6523.59 | 6096.91 | 153.66 | 6544.29 |
| **Case-3** | $\Delta x, m$ | 0.44 | 0.29 | 1.42 | 0.18 | 0.06 | 0.59 |
| | $\Delta\lvert V\rvert, \frac{m}{s}$ | 1.36 | 1.29 | 4.01 | 0.14 | 0.24 | 3.15 |
| | $\Delta\theta, ^\circ$ | 0.64 | 0.51 | 2.89 | 0.14 | 0.13 | 2.16 |
| | $\Delta m_{fuel}, kg$ | 6083.3 | 143.87 | 6515.08 | 6100.95 | 152.79 | 6521.85 |
| **Case-4** | $\Delta x, m$ | 0.45 | 0.3 | 1.41 | 0.19 | 0.07 | 1.13 |
| | $\Delta\lvert V\rvert, \frac{m}{s}$ | 1.37 | 1.36 | 4.91 | 0.21 | 0.42 | 5.04 |
| | $\Delta\theta, ^\circ$ | 0.65 | 0.53 | 2.64 | 0.18 | 0.26 | 3.94 |
| | $\Delta m_{fuel}, kg$ | 6082.37 | 146.19 | 6566.52 | 6094.90 | 146.21 | 6436.72 |
| **Case-5** | $\Delta x, m$ | 0.36 | 0.27 | 1.72 | 0.29 | 0.26 | 1.12 |
| | $\Delta\lvert V\rvert, \frac{m}{s}$ | 3.53 | 1.53 | 6.15 | 1.63 | 1.88 | 6.52 |
| | $\Delta\theta, ^\circ$ | 1.28 | 0.61 | 3.04 | 0.75 | 0.62 | 3.54 |
| | $\Delta m_{fuel}, kg$ | 6029.43 | 166.04 | 6510.21 | 6076.75 | 184.21 | 6514.69 |
| **Case-6** | $\Delta x, m$ | 0.54 | 0.5 | 4.24 | 0.41 | 0.37 | 1.15 |
| | $\Delta\lvert V\rvert, \frac{m}{s}$ | 3.64 | 3.18 | 8.73 | 1.57 | 1.82 | 6.76 |
| | $\Delta\theta, ^\circ$ | 1.36 | 1.09 | 4.59 | 1.53 | 1.26 | 4.24 |
| | $\Delta m_{fuel}, kg$ | 6021.59 | 175.03 | 6543.35 | 6083.41 | 185.09 | 6515.03 |

catastrophically. The position errors were still within the acceptable range. Nevertheless, our Monte-Carlo analysis showed that the touch-down velocities increased with the addition of measurement noise. This is attributed to the bang-bang nature of the policies since the additional noise affects the switching times at some points of the simulations, which causes these unwanted increases in landing velocities. However, when the magnitudes are considered, we assume that they can be tackled with extra effort in the last few seconds of the landing.

## V. DISCUSSION

In our study, we effectively integrated optimal control and learning control approaches, harnessing their individual strengths to overcome the limitations of non-learning and non-optimal methods. Our real-time guidance policy was formulated using neural networks, which possess an enhanced learning capacity. These networks were trained to obtain the optimal control solution, specifically targeting the minimization of fuel consumption. Consequently, our approach offers distinct advantages over classical guidance algorithms, effectively reducing operational costs. Moreover, unlike non-learning guidance algorithms, particularly convexification approaches, our method leverages the power of parametric modeling. This enables the execution of guidance commands in each step, allowing for the re-computation of optimal actions without resorting to oversimplifications or compromises on the original problem. This adaptability and flexibility provide robustness in handling off-nominal conditions, a significant advantage of the proposed approach. The simulation results presented in this study provide empirical evidence to support our claims. They unequivocally demonstrated that our optimized policy achieves robust performance, effectively addressing the uncertainties encountered during vehicle steering along a fuel-optimal trajectory. Finally, to assess the real-time implementation applicability of our optimized models, we embedded them in a single-board computer. Through rigorous testing, we measured the inference times of our models, revealing that they could execute each guidance step within microseconds. This measurement further emphasizes the feasibility and practicality of real-time implementation.

In a recent study [8], the authors proposed a real-time guidance policy named DCRNG (Deep Classification and Regression Network-based Guidance), which employs two DNNs to establish a nonlinear mapping between the ideal state and control pairings for a 3-degree-of-freedom motion model similar to the framework used in our study. (While they utilized three translational kinematic equations, we opted for two translational kinematic equations combined with one rotational dynamic equation.) One DNN was responsible for the problem classification, whereas the other was employed for regression to generate guidance commands. Upon examining their results, it is apparent that the authors employed DNNs with significantly larger network architectures than those in our approach. Specifically, they used networks with 10 hidden layers, each consisting of 256 neurons, for both networks. This results in a total parameter count that is 2000 times higher than that of our best-performing models. The computational complexity and potential overfitting challenges that arise in such high-dimensional networks are worth noting. In addition, our models are more suitable for real-time computations and real-world implementations owing to their lightweight memory requirements. Furthermore, the authors presented Monte Carlo analysis results considering uncertainties in the drag force ranging from 1% to 5%. Their findings indicated a correlated increase in the mean position and velocity errors as the uncertainty level increased. The generalization capability of their trained networks may be limited as they struggle to adapt appropriately to novel scenarios (Out-of-distribution, OOD). However, we conducted an extensive Monte Carlo analysis in our study, which encompasses a broader range of factors beyond aerodynamic uncertainty. The results demonstrate that the proposed guidance policy exhibits remarkable adaptability to novel conditions and out-of-domain distributions. This highlights the robustness and generalization capabilities of our approach compared to the limitations of the DCRNG method.

In [23], the authors integrated imitation learning with deep reinforcement learning techniques to formulate a guidance policy for the reusable rocket landing problem within a 3-degree-of-freedom motion model, similar to our study. In the imitation learning phase, the authors initially trained a guidance policy network using data obtained from optimal control problem solutions. Subsequently, they applied proximal policy optimization within the reinforcement learning framework to improve the performance of their guidance policy. Initially, only the critic (state value) network was updated, and after a certain number of epochs, the actor (guidance) network was also updated. Through this combined training methodology, the authors achieved a mean of 6 % higher fuel consumption compared to fuel-optimal trajectories obtained from GPOPS software over 100 trajectory samples. In contrast, our proposed approach produced trajectories that are nearly optimal in terms of fuel efficiency. Additionally, our Monte Carlo analysis illustrates the exceptional robustness of our models, maintaining high accuracy under various disturbances–a characteristic not distinctly emphasized in the comparative study.

Regarding the stability analysis, in our proof-of-concept study, we assumed that aerodynamic moments could be neglected compared to the moments stemming from thrust control. Consequently, examining the stability of a system with rigorous proof is challenging. To better understand the effect of the guidance system on the overall control regime and vehicle stability, we performed extensive Monte Carlo analysis. Monte Carlo analysis involved simulating multiple scenarios with varying parameters, disturbances, and uncertainties to test the effect of the proposed guidance system on its stability and robustness. Our proposed approach has no negative effect on the system's stability, as our guidance command generator utilizes system dynamics and physical limitations. However, since the booster system is unstable in the open loop, feedback control is required for stabilization. In our study, we did not implement any stabilizing controllers in our simulations, which resulted in oscillations in the system's responses. Nevertheless, oscillatory responses can be observed depending on the degree of instability, even with feedback control. An analogous example is the Pole-cart problem in control engineering, which shares dynamic similarities with an aerodynamically unstable rocket [49]. The transient responses of the angular position and angular acceleration of the pendulum exhibit oscillatory behavior before reaching a steady state despite being controlled by feedback laws.

A few suggestions are considered for further research. We will consider applying the scientific machine learning method with our proposed neural network models to high-fidelity simulations like the 6-degree-of-freedom motion model with more realistic subsystems such as the aerodynamic and actuator models. However, when realistic subsystem models are considered, they may be non-differentiable. In this case, owing to the learning outcomes gained from this proof-of-concept work, we consider using data-driven surrogate models, which enable it to be differentiable, making it feasible to employ the proposed methodology. Moreover, as discussed in our study, we did not implement a reference tracker or stabilizing controller in our simulations. In future work, we plan to implement an inner loop controller in our guidance policy optimization loop to better assess the system's overall stability and guidance and control loop interactions. Finally, to further boost the performance, we will consider implementing the outstanding self-adaptive loss weights concept to our hand-crafted loss function, which was initially proposed for physics-informed neural networks [50]. Hence, by eliminating the grid-search approach for the hyper-parameter of the loss weights, obtaining a more promising training performance might be possible.

## VI. CONCLUSION

In this pivotal study, we have advanced reusable rocket landing guidance by integrating scientific machine learning with novel neural network architectures, specifically Adaptive Quadratic Residual Neural Networks (AQResNet) and Rowdy Adaptive Quadratic Residual Neural Networks (RAQResNet). The application of universal ordinary differential equations within this framework has proven instrumental in reducing prediction uncertainty and the amount of training data required, while also enhancing convergence speed and accuracy, particularly in challenging out-of-domain scenarios.

The simulation results show that our proposed approach with two novel NN models can learn from scarce data and physical laws simultaneously. As a result, an enormous performance boost was observed, as it reduced uncertainty and improved extrapolation and out-of-domain prediction capabilities. Indeed, Monte Carlo analysis showed that our trained network's performance was unaffected by the disturbances and uncertainties. This reveals how the trained policy models generalize well to novel conditions that are not experienced during optimization by learning the governing physics. In addition, experiments demonstrated that our proposed architectures have much more expressive power than the vanilla neural network architectures. To illustrate, the

RAQResNet model achieved approximately 300 times lower validation loss than the vanilla architecture with the same number of trainable parameters. This quantifiable improvement highlights the transformative impact of our approach in dealing with complex neural network architectures.

Moreover, when real-world implementation is considered, the computational loads of our offline-trained network models are so low that they only include three low-level matrix multiplications. Moreover, the practical applicability of these models has been underscored by real-world implementation tests. Our models demonstrated impressive performance on a Raspberry Pi 4, with inference times recorded in the range of several microseconds, showcasing their potential for real-time computations in modern flight hardware. This efficiency in computational speed is critical for aerospace applications, where rapid and reliable decision-making is paramount.

In summary, our research significantly contributes to the field of aerospace engineering by enhancing the accuracy and reliability of guidance systems for reusable rocket landings. The methodologies and findings presented not only provide a robust framework for future aerospace applications but also lay the groundwork for further exploration in more complex simulations and control systems, leveraging the strengths of scientific machine learning and advanced neural network architectures.

## REFERENCES

[1] M. Szmuk, B. Acikmese, and A. W. Berning, "Successive convexification for fuel-optimal powered landing with aerodynamic drag and non-convex constraints," in *Proc. AIAA Guid., Navigat., Control Conf.*, 2016, pp. 378–394.

[2] A. Botelho, M. Martinez, C. Recupero, A. Fabrizi, and G. De Zaiacomo, "Design of the landing guidance for the retro-propulsive vertical landing of a reusable rocket stage," *CEAS Space J.*, vol. 14, no. 3, pp. 551–564, Jul. 2022.

[3] J. Hwang and J. Ahn, "Integrated optimal guidance for reentry and landing of a rocket using multi-phase pseudo-spectral convex optimization," *Int. J. Aeronaut. Space Sci.*, vol. 23, no. 4, pp. 766–774, Sep. 2022.

[4] Z. Song, C. Wang, S. Theil, D. Seelbinder, M. Sagliano, X. Liu, and Z. Shao, "Survey of autonomous guidance methods for powered planetary landing," *Frontiers Inf. Technol. Electron. Eng.*, vol. 21, no. 5, pp. 652–674, 2020.

[5] M. Gallaher, D. Coughlin, and D. Krupp, "A guidance and control assessment of three vertical landing options for RLV," in *Proc. Guid., Navigat., Control Conf.*, 1995, p. 3702.

[6] Y. Ishijima, S. Matsumoto, and K. Hayashi, "Re-entry and terminal guidance for vertical-landing tsto (two-stage to orbit)," in *Proc. Guid., Navigat., Control Conf. Exhibit*, 2012, pp. 192–200.

[7] M. Sagliano, "Generalized hp pseudospectral-convex programming for powered descent and landing," *J. Guid., Control, Dyn.*, vol. 42, no. 7, pp. 1562–1570, Jul. 2019.

[8] J. Wang, H. Ma, H. Li, and H. Chen, "Real-time guidance for powered landing of reusable rockets via deep learning," *Neural Comput. Appl.*, vol. 35, no. 9, pp. 6383–6404, Mar. 2023.

[9] L. Blackmore, "Autonomous precision landing of space rockets," in *Proc. Frontiers Eng., Rep. Lead.-Edge Eng. Symp.*, Washington, DC, USA, vol. 46, 2016, pp. 15–20.

[10] M. Sagliano, T. Tsukamoto, J. A. M. Hernández, D. Seelbinder, S. Ishimoto, and E. Dumont, "Guidance and control strategy for the CALLISTO flight experiment," in *Proc. 8th EUCASS Conf. Aeronaut. Aerosp. Sci.*, 2019, pp. 1–13.

[11] P. Simplício, A. Marcos, and S. Bennani, "Guidance of reusable launchers: Improving descent and landing performance," *J. Guid., Control, Dyn.*, vol. 42, no. 10, pp. 2206–2219, Oct. 2019.

[12] B. Gaudet and R. Furfaro, "A comparison of partially and fully integrated guidance and flight control optimized with reinforcement meta-learning," in *Proc. AIAA SCITECH Forum*, 2023, p. 1628.

[13] M. Szmuk and B. Acikmese, "Successive convexification for 6-DoF Mars rocket powered landing with free-final-time," in *Proc. AIAA Guid., Navigat., Control Conf.*, 2018, p. 0617.

[14] R. Chai, A. Tsourdos, A. Savvaris, S. Chai, Y. Xia, and C. Philip Chen, "Review of advanced guidance and control algorithms for space/aerospace vehicles," *Prog. Aerosp. Sci.*, vol. 122, Apr. 2021, Art. no. 100696.

[15] T. M. A. Habib, "Artificial intelligence for spacecraft guidance, navigation, and control: A state-of-the-art," *Aerosp. Syst.*, vol. 5, no. 4, pp. 503–521, Dec. 2022.

[16] D. Izzo, M. Märtens, and B. Pan, "A survey on artificial intelligence trends in spacecraft guidance dynamics and control," *Astrodynamics*, vol. 3, no. 4, pp. 287–299, Dec. 2019.

[17] L. Cheng, Z. Wang, Y. Song, and F. Jiang, "Real-time optimal control for irregular asteroid landings using deep neural networks," *Acta Astronautica*, vol. 170, pp. 66–79, May 2020.

[18] Y. Song, X. Miao, L. Cheng, and S. Gong, "The feasibility criterion of fuel-optimal planetary landing using neural networks," *Aerosp. Sci. Technol.*, vol. 116, Sep. 2021, Art. no. 106860.

[19] C. Sánchez-Sánchez and D. Izzo, "Real-time optimal control via deep neural networks: Study on landing problems," *J. Guid., Control, Dyn.*, vol. 41, no. 5, pp. 1122–1135, May 2018.

[20] R. Furfaro, I. Bloise, M. Orlandelli, P. Di Lizia, F. Topputo, and R. Linares, "Deep learning for autonomous lunar landing," *Adv. Astron. Sci.*, vol. 167, pp. 3285–3306, Mar. 2018.

[21] L. Cheng, Z. Wang, F. Jiang, and J. Li, "Fast generation of optimal asteroid landing trajectories using deep neural networks," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 56, no. 4, pp. 2642–2655, Aug. 2020.

[22] W. Li, Y. Song, L. Cheng, and S. Gong, "Closed-loop deep neural network optimal control algorithm and error analysis for powered landing under uncertainties," *Astrodynamics*, vol. 7, no. 2, pp. 211–228, Jun. 2023.

[23] L. Su, J. Wang, Z. Ma, and H. Chen, "Real-time guidance for powered landing of reusable rockets via deep reinforcement learning," in *Proc. IEEE Int. Conf. Unmanned Syst. (ICUS)*, Oct. 2022, pp. 214–219.

[24] L. Nugroho, R. Andiarti, R. Akmeliawati, A. T. Kutay, D. K. Larasati, and S. K. Wijaya, "Optimization of reward shaping function based on genetic algorithm applied to a cross validated deep deterministic policy gradient in a powered landing guidance problem," *Eng. Appl. Artif. Intell.*, vol. 120, Apr. 2023, Art. no. 105798.

[25] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, A. Ramadhan, and A. Edelman, "Universal differential equations for scientific machine learning," 2020, *arXiv:2001.04385*.

[26] L. Huang, D. Vrinceanu, Y. Wang, N. Kulathunga, and N. R. Ranasinghe, "Discovering nonlinear dynamics through scientific machine learning," in *Intelligent Systems and Applications*. Switzerland: Springer, 2022, pp. 261–279.

[27] S. Beregi, D. A. W. Barton, D. Rezgui, and S. Neild, "Using scientific machine learning for experimental bifurcation analysis of dynamic systems," *Mech. Syst. Signal Process.*, vol. 184, Feb. 2023, Art. no. 109649.

[28] J. Bu and A. Karpatne, "Quadratic residual networks: A new class of neural networks for solving forward and inverse problems in physics involving PDEs," in *Proc. SIAM Int. Conf. Data Mining (SDM)*, Philadelphia, PA, USA: SIAM, 2021, pp. 675–683.

[29] M. U. Demirezen, "Quadratic residual multiplicative filter neural networks for efficient approximation of complex sensor signals," *IEEE Access*, vol. 11, pp. 75236–75268, 2023.

[30] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville, "On the spectral bias of neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 5301–5310.

[31] S. Fridovich-Keil, R. G. Lopes, and R. Roelofs, "Spectral bias in practice: The role of function frequency in generalization," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 35, 2022, pp. 7368–7382.

[32] A. D. Jagtap, K. Kawaguchi, and G. Em Karniadakis, "Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks," *Proc. Roy. Soc. A*, vol. 476, no. 2239, 2020, Art. no. 20200334.

[33] A. D. Jagtap, K. Kawaguchi, and G. E. Karniadakis, "Adaptive activation functions accelerate convergence in deep and physics-informed neural networks," *J. Comput. Phys.*, vol. 404, Mar. 2020, Art. no. 109136.

[34] A. D. Jagtap, Y. Shin, K. Kawaguchi, and G. E. Karniadakis, "Deep Kronecker neural networks: A general framework for neural networks with adaptive activation functions," *Neurocomputing*, vol. 468, pp. 165–180, Jan. 2022.

[35] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *J. Comput. Phys.*, vol. 378, pp. 686–707, Feb. 2019.

[36] U. Çelik, "Robust booster landing guidance/control," M.S. thesis, Optim. Syst. Theory, KTH, 2020.

[37] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, New York, NY, USA, 2016, pp. 785–794.

[38] A. Vakayil and V. R. Joseph, "Data twinning," *Stat. Anal. Data Mining: ASA Data Sci. J.*, vol. 15, no. 5, pp. 598–610, 2022.

[39] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min.*, 2019, pp. 2623–2631.

[40] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, 1989.

[41] T. Chen and H. Chen, "Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems," *IEEE Trans. Neural Netw.*, vol. 6, no. 4, pp. 911–917, Jul. 1995.

[42] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.

[43] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, *arXiv:1609.04747*.

[44] M. M. Lau and K. H. Lim, "Review of adaptive activation function in deep neural network," in *Proc. IEEE-EMBS Conf. Biomed. Eng. Sci. (IECBES)*, Dec. 2018, pp. 686–690.

[45] X. Jin, C. Xu, J. Feng, Y. Wei, J. Xiong, and S. Yan, "Deep learning with S-shaped rectified linear activation units," in *Proc. 13th AAAI Conf. Artif. Intell.*, 2016, pp. 1737–1743.

[46] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Rev.*, vol. 59, no. 1, pp. 65–98, Jan. 2017.

[47] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent.*, Dec. 2014, pp. 1–15.

[48] C. Sanderson and R. Curtin, "Armadillo: A template-based C++ library for linear algebra," *J. Open Source Softw.*, vol. 1, no. 2, p. 26, Jun. 2016.

[49] J. Pei and P. Rothhaar, "Demonstration of the space launch system augmenting adaptive control algorithm on pole-cart platform," in *Proc. AIAA Guid., Navigat., Control Conf.*, 2018, p. 0608.

[50] L. D. McClenny and U. M. Braga-Neto, "Self-adaptive physics-informed neural networks," *J. Comput. Phys.*, vol. 474, Feb. 2023, Art. no. 111722.

**UGURCAN ÇELIK** received the Graduate degree from the Aerospace Engineering Department, Middle East Technical University, one of Turkey's leading universities, in 2017, and the master's degree from the KTH Royal Institute of Technology, Sweden. He is currently pursuing the Ph.D. degree with Cranfield University under the supervision of Prof. Gokhan Inalhan. Following his graduation, he began working as a System Design Engineer with ROKETSAN Missiles Inc., a prominent company in the defense industry. After a year of employment, he embarked on a journey to further his education. During the master's degree, he gained valuable experience in the field of artificial intelligence. Upon completing his education, he returned to Turkey and continued to advance his career in the realm of artificial intelligence.

**MUSTAFA UMUT DEMIREZEN** (Senior Member, IEEE) received the B.S., M.Sc., and Ph.D. degrees in electrical and electronics engineering from Gazi University, in 2000, 2002, and 2015, respectively. From 2000 to 2013, he was an Engineer Officer with the Turkish Naval Forces, from 2000 to 2013. After resigning from the Turkish naval forces, he worked for several defense companies, such as STM Defence Inc. and ROKETSAN Missiles Inc. as a Researcher and then a Group Manager for research and development projects in artificial intelligence, from 2013 to 2021. He was with the Huawei Research and Development Center as a Research and Innovation Manager. Since 2023, he has been with UDemy Inc. as a Senior Manager in machine learning engineering with the Data Products Department. He is the author of five artificial intelligence subjected books and more than 20 scientific articles. His research interests include artificial intelligence, deep learning, neuromorphic computing, spiking neural networks, scientific machine learning, and physics-informed neural networks. He is an Associate Editor of the journal of *Mesasurement and Control*. He was a recipient of the AIAA/IEEE Digital Avionics Systems Conference (DASC) Best Session Paper Award, in 2021, and IEEE UBMK-2022, International Conference On Computer Science and Engineering, Best Paper (Third place) Award, in 2022.