An Autonomous Reinforcement Learning Framework for Fault Recovery and Mission Replanning on CubeSats

Ganesh Danke*, Rajiv Iyer*
Northeastern University
360 Huntington Ave, Boston, MA 02115
danke.g@northeastern.edu, iyer.raj@northeastern.edu

*all authors contributed equally

Faculty Advisor: Brian Hulbert
Northeastern University
hulbert.b@northeastern.edu

ABSTRACT

Onboard system failures during CubeSat operation can have significant consequences for mission success. Limited resources during the development process can hamper the development and implementation of recovery systems, increasing the likelihood of mission failures. In response, this paper establishes a reusable autonomous framework for mission replanning in the event of an onboard system failure. Prior to launch, the framework ingests a standardized mission plan detailing mission objectives, mission priorities, and onboard capabilities and resources. Segmenting this information into a set of discrete tasks with completion dependencies, a reinforcement learning approach is used to select a schedule of tasks with the greatest priority while meeting resource limitations. This selection is scheduled into a new mission plan using a modified reinforcement learning approach. Testing this framework on a series of simulated satellite missions, it demonstrates moderate success in adapting multi-system failures, such as a variety of attitude control, power storage and generation, and computational faults.

INTRODUCTION

CubeSat missions have historically been prone to failure. From 2000 to 2015, only 17.4% of 288 planned CubeSat missions resulted in a full success, with 67.7% of CubeSats that reached orbit experiencing some failure. A variety of proposed factors exist to explain the frequency of these failures, including a lack of robust development and testing expertise, limited budgets and resources, and ambitious mission designs.[1] In a study conducted at the CalPoly CubeSat workshop, Alanazi and Straub found that 48% of respondents experienced tool failures, 24% experienced failures from model usage, and the remaining 28% experienced failures due to both factors - demonstrating that there is diversity when it comes to errors in small satellites. Corroborated by a survey of CubeSat developers indicating that faster testing procedures contribute to mission success, the ability to easily identify, predict, and adapt to a variety of issues is crucial to reducing mission failures.[3] While some failures may be detectable on the ground, it is notable that 62.6% of in-flight failures occurred during the mission while the CubeSat was previously operational, rather than an already dead or flawed unit being launched.[1] By implementing systems which can detect these unforeseen in-flight faults and develop recovery strategies in real-time, mission failures can be mitigated without placing significant additional burdens on CubeSat developers to model and predict such faults.

A few approaches exist for the problem of CubeSat fault detection and recovery. One such approach is outlined by Rao - in which battery current and voltage is used as a “root of trust” in comparison with metrics from onboard sensors and systems on a specific CubeSat, training a model. Divergences between realtime values and the model, such as during onboard failures, are flagged as anomalies, enabling the detection of unforeseen faults in a highly flexible manner.[2] In another approach, Lobo establishes an architecture for detecting and isolating faults on CubeSats in which hardware redundancy is utilized, defining this process in detail for a number of specific onboard systems. This enables effective fault mitigation during small-scale failures.[6] While such solutions demonstrate efficacy on real-world CubeSat systems, they have a few key limitations. These solutions necessitate a high degree of human involvement, with the former requiring manual analysis and mission plan rescheduling following fault detection, and the latter requiring highly system specific mitigation strategies to be implemented. The reliance on redundancy for the

latter approach provides minimal opportunities for recovery during large-scale faults. Additionally, both models focus on the short term consequences of faults rather than the impact on longer term mission schedules and objectives.

Given a gap in flexible autonomous recovery following the detection of faults, the development of a standardized, reusable framework for fault recovery will have the greatest impact on mitigating preventable in-flight mission failure. To ensure that fault mitigations better address long-term mission objectives and their interactions, this framework will take the rearrangement of CubeSat task schedules per orbital period. Existing scheduling work by Liang demonstrates that greedy heuristic approaches provide greater flexibility and scalability than dynamic programming or genetic algorithms, albeit with some minimal reductions in accuracy.[4] Thus, this framework will take the form of a greedy reinforcement learning algorithm, leveraging the known mission objectives preflight to reduce real time processing complexity during faults.

FRAMEWORK DESIGN

Overview

The fault detection framework is based around the training and development of two reinforcement learning models based on the initial mission plan of the CubeSat, autonomously generating a new mission schedule during a fault. From the initial mission plan, the first model selects an optimal subset of tasks: prioritizing mission-critical tasks that fit into time and onboard resource restrictions. Subsequently, the second model assembles these tasks into a new mission plan , aligning as close as possible with target times and locations. These models are trained in tandem, ingesting a series of simulated faults applied upon the initial mission plan. For the purposes of the initial development of the framework, tasks are considered to be independent and faults consist of reductions to the resources of various onboard systems. This approach utilizes a technique of model caching, enabling the usage of known mission and system information to decrease the computational strain of running the framework during a fault, rather than performing this calculation entirely at runtime. By eliminating having to relearn tasks and retrain the model post-development, the ease and speeds of fault response and adaptation are greatly enhanced. The following diagram illustrates the analysis and training pipeline utilized to generate these reinforcement learning models before flight, and their application following an onboard fault during a mission.

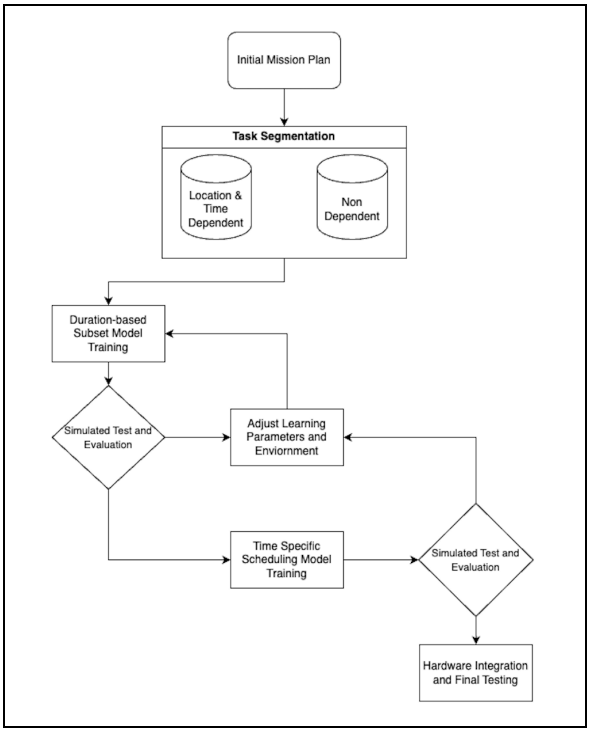

Figure 1: Framework Design

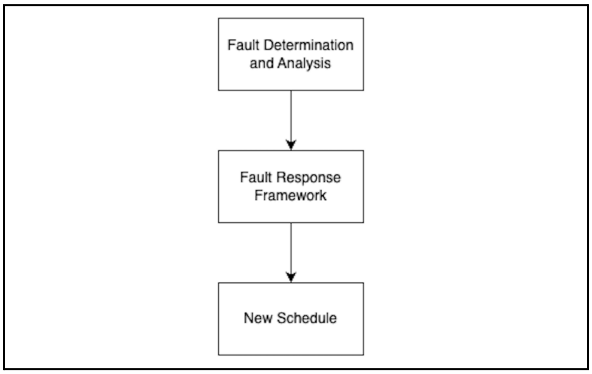

Figure 2: Framework Response

Initial Mission Plan

A standardized initial mission plan is utilized as the basis for the fault recovery strategy. This ensures that sufficient information regarding onboard capabilities and mission objectives is provided to generate viable and optimal task schedules. To support both human configurability and automated processing and generation, this mission plan will be structured as a YAML document representing a single orbital period. It consists of two primary sections: a description of onboard systems and their resource limitations, and a sequential schedule of mission objectives and their

resource dependencies. Onboard systems are defined using a name and a series of parameters describing the available resources on the system and applicable limitations. For example, a magnetorquer may have limitations on maximum current draw. Mission objectives are defined using a name, the system(s) it utilizes, the quantity of resources utilized on these systems, the location or time frame in which an objective must be completed if applicable, the duration of time required to complete the objective, and the priority of an objective. For example, a communication task may need to be completed over a ground station at a specific time, whereas a systems diagnostic check can be conducted at any point. Combined, this information describes the resources available to the satellite in ideal conditions and how they are utilized throughout the mission.

```
onboard-systems:
  - id: "FEEP ADCS"
    resource-quantity: "15 ml"
    task-access-limit: 1
    concurrent-strain: "3 ml"
  - id: "Onboard Battery Power"
    resource-quantity: "12 watt-hours"
    task-access-limit: 4
    concurrent-strain: "1 watt-hours"

mission-schedule:
  - id: "Detumbling"
    position: [-25.5548, -19.2973, 419861]
    timestamp: 2024-04-27T15:30:00Z
    priority: 5
    duration: "60 s"
    resources:
      system: "FEEP ADCS"
      consumption: "2 ml"
  - id: "Extend Solar Panels"
    position: [-25.5540, -19.2960, 419861]
    timestamp: None
    priority: 7
    duration: "30 s"
    resources:
      system: "Onboard Battery Power"
      consumption: ".1 watt-hours"
```

Figure 3: Basic Mission Plan Configuration

To quantify the optimality of a particular schedule, the priority of tasks and objectives is defined by a numerical weightage. For simplicity, this weightage uses a scale from 1 to 10, with 1 indicating lowest priority and 10 indicating highest priority. This priority value is used as the basis for the reward scheme utilized in the reinforcement learning approach, influencing the generated schedule. A schedule is considered to be more optimal than another if it has a higher total weightage while respecting the resource and capability limitations of the CubeSat. In ideal conditions, the initial schedule is considered to be the most optimal schedule as it contains all desired tasks - though realistically, there are limitations. This system provides greater flexibility in mission replanning, compared to a strictly ordered hierarchy of tasks, by enabling a greater variety of valid task combinations.

Task Classification

The initial mission plan is processed into a series of atomic tasks, with their respective system and resource requirements, weightages, and completion conditions. For example, a larger mission objective of taking a photograph at the periapsis of the orbit may be broken down into the tasks of detumbling, extending solar panels, using the attitude control systems to orient the camera to be nadir-facing, and finally taking a picture. Tasks are then further segmented into tasks with strict location and/or time dependencies, and general tasks without these dependencies.

Tasks with these dependencies must be completed at a particular location or time in orbit to be considered successful. An example of such a task is taking a photograph of the earth at the apoapsis of an orbit, using a long exposure time. Starting this task too late or too early fails the necessary criteria of being at apoapsis; thus, if such a task is included in a rescheduling, it would always be placed as close as possible to its target time or location. Non-dependent tasks can be executed at any point in orbit. An example of such a task is having to perform an intensive photo compression task that takes 30 minutes to complete - it takes up a portion of the schedule but does not require a specific time or location. These tasks can be flexibly moved within the mission schedule, but may occupy an extended period of time. Overall, tasks must be scheduled such that they do not conflict or overlap with one another.

MODEL I - DURATION-BASED SUBSET

Following a system fault, sufficient resources may not exist to execute all of the tasks within the initial mission plan. To ensure the efficacy of the fault recovery strategy, a subset of the tasks from the initial mission plan must be generated such that resource and time limitations are satisfied whilst maximizing the priority of the subset. This selection problem can be modeled as whether or not to select some task x given the current remaining onboard resources y, iterating over the total set of tasks X for the current orbital period given an initial set of resources Y. Within each iteration, the resource cost of the current task, the current remaining

resources, and the priority of the task serve as concrete metrics to the viability of selecting a particular task.

A Deep-Q neural network (DQN) was selected to model this network due to the advantages it provides within this context. The model performs well when provided with a well defined action set, observation space, and reward description such as in this task selection problem. While other approaches may tolerate more ambiguous specifications, the ability to rely on these concrete definitions gives the DQN model the added advantage of low computational costs. This decreases training time, allowing greater agility in development, and also makes the model suitable for low power satellite onboard computers (OBCs).[5] These specifications are structured in an environment: a structure that handles updating the state of the problem following the choice made in each iteration, and providing the model with the resultant observation.

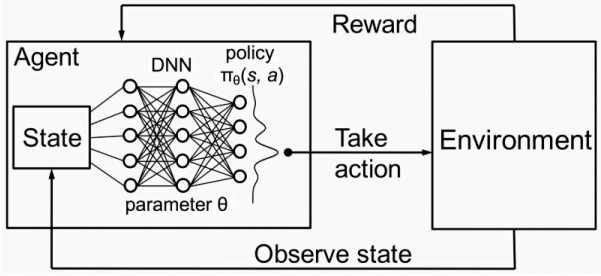

Figure 4: The processes of a Deep-Q neural network[7]

The environment for this problem was constructed as follows. The reward scheme for task scheduling is based solely on the priority value of the task, such that the valid selection of higher priority tasks produces greater rewards. The action space of this network is 1 and 0: choose to add the task to the subset, or choose not to add the task to the subset. The observation space is defined as an array of four integers representing the resource cost of the current task, the priority of the current task, the remaining quantity of the onboard resource the task uses, and the time duration occupied by the task. The algorithm continues iterating until a decision of inclusion or exclusion has been made on every task within the initial mission plan. After this end condition is reached, the algorithm outputs a subset of tasks taken from the initial list of specified tasks.

MODEL II - TIME-SPECIFIC SCHEDULING

After selecting a set of optimal tasks, these tasks must be organized into a schedule with assigned start times to produce an actionable new mission plan. The primary objective of schedule generation is to fit all tasks from the set into the schedule without overlap while respecting tasks that have specific target times/locations. This selection problem can thus be modeled as choosing whether or not to schedule a task x at a time y, iterating over both set X representing the tasks to be scheduled and set Y representing the time intervals tasks can be scheduled on. Clear metrics exist to evaluate the efficacy of each such choice: the priority of the current task and the proximity to the target time (if applicable). Thus, due to the discrete action space and well defined observation per each iteration of this problem, a Deep-Q neural network approach is suitable for re-application in this problem.

The environment for this network was constructed as follows. The reward scheme for scheduling is segmented based on the categorization of a task. Time/location-dependent tasks affect the reward via an exponential relationship based on the proximity of their scheduled time to their target time - the closer the proximity, the higher the reward. Non-dependent tasks are rewarded their constant priority if they are scheduled at any point. The action space provided to the network is 1 and 0: choose to schedule the task at that time, or choose not to schedule the task at that time. The observation space is the same as the previous model, with the cost of the current task, the priority of the task, the remaining resources relevant to the task, and the task duration. The algorithm continues iterating until all tasks in the subset are scheduled. After this end condition is reached, this algorithm outputs a finalized schedule consisting of tasks mapped to specific execution times.

SIMULATED TESTING

Methodology

An experiment-based approach was utilized to test the efficacy of the fault recovery framework. For the purpose of this testing, faults are represented by one or multiple of a CubeSat's onboard systems losing a significant quantity of resources, such as an ADCS system losing a propellant tank. Each experimental trial consisted of training the framework on a randomly generated initial mission plan. Prior to training, system faults were randomly applied to the mission plan in varying degrees of severity to ensure that the models developed adequate adaptability. To test the efficacy of the model for use in fast development and to analyze the lower bound of its performance, only 4000 training iterations were conducted for the two models per trial with training taking an average of 5 minutes or less for both of the models combined. For consistency, each mission plan consisted of the three systems of ADCS, power, and computation each with a numerical quantity

of resources, and contained 50 distinct tasks. The framework was then tasked with rescheduling this initial plan 30 times given a series of faults. To ensure the reproducibility and validity of these results, 20 different trials were conducted on a Google Colab L4 GPU instance, totaling 600 individual results.

The results of these trials, and thus the quality of the resulting schedules, was analyzed by measuring how close the reward of the generated plan is to the reward of the most optimal schedule (generated via a top-down dynamic programming approach) for the associated initial plan and system fault. For example, if the framework outputs a schedule with a priority of 80 points but the optimal schedule has 100 points, the efficacy of the framework in that instance is 80%. Thus, close scores would indicate that the model was effective at calculating the optimal solution, whereas distant scores would indicate that the model performed poorly at this task. Other metrics, such as the variability of the results in relation to the severity of the fault, and the speed of the framework in comparison to a dynamic programming approach are analyzed to extract potential insight into the specific failings of the model, as well as some potential ways to address them. These metrics, used in conjunction, provide a sufficient analysis of the effectiveness of our solution alongside avenues for improvement.

Results

After performing 20 trials of fault recovery, the results were analyzed as previously described to determine the efficacy of the framework. Across 600 schedules, the mean and median efficacy were 72.73% and 77.91%, with 170 instances of the framework generating the optimal schedule. The standard deviation and interquartile range of this distribution is 26.84% and 46.18% respectively. These results indicate that while the framework is moderately effective at generating the optimal solution even with minimal training, there are instances in which it generates schedules of very poor quality. In one instance, the schedule generated by the framework had 0% of the efficacy of the optimal solution. In a real world mission, this scenario would effectively constitute a failure. This distribution is illustrated by the following histogram, which demonstrates the leftwards skew of the distribution.

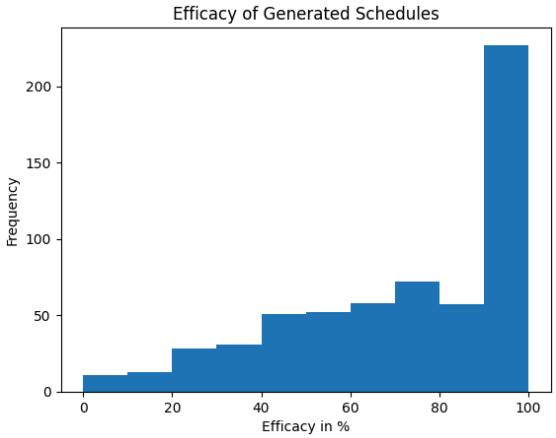

Figure 5: Efficacy of Generated Schedules

A potential explanation for such results is the low number of iterations used in the training process for the two models combined with a suboptimal observation specification for the second model. Thus, the second model would require a greater number of training iterations in order to achieve the desired level of generalizability. This premise is supported by an analysis of fault severity in comparison to model efficacy as a similarity score. The following figure maps raw fault severity, the total difference in onboard resources between the initial mission plan and a simulated fault across all systems, against the efficacy of the schedule generated following that fault.

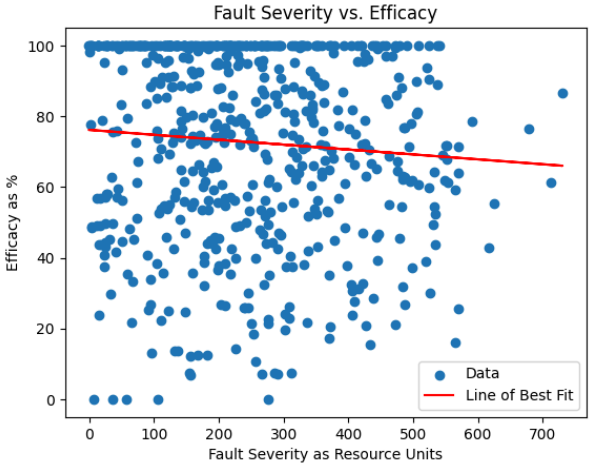

Figure 6: Fault Severity vs. Efficacy

As demonstrated by the best fit line, which has a r-squared value of 0.6%, there exists a low degree of correlation between the severity of a fault and the ability of the framework to effectively address such a fault. In comparison, the first model has a mean efficacy of 96.46% with a standard deviation of 7.97%, indicating that it was more effective at utilizing the

training data to develop an effective policy irrespective of fault severity. Thus, by changing the observation specification and iteration structure of the second model, it may be possible to improve the overall efficacy of the framework. This could include implementing the distance from the target time of the current task within the observation spec, or changing the scheduling strategy to always force location dependent tasks to occur at the target times.

To determine the real-time effectiveness of the framework, the speed of its schedule generation was compared against that of the optimal DP solution. The runtime of the framework had a mean of 0.65 seconds with a standard deviation of 0.07 seconds, while the runtime of the DP approach had a mean of 4.77 seconds and a standard deviation of 2.12 seconds. This demonstrates that the model caching technique used in the design of the framework is effective at reducing the real-time runtime. The measurable variation for both approaches can be explained by the variable number of tasks which are selected for the final schedule: less tasks to schedule results in a faster time to schedule them. The larger variability of the DP approach can be attributed to its inherent poor scalability, as its runtime is impacted equally by the number of tasks, and the resource capacities of the system following a fault. In comparison, the framework is largely only impacted by the number of tasks, which controls the number of iterations the DQN model completes.

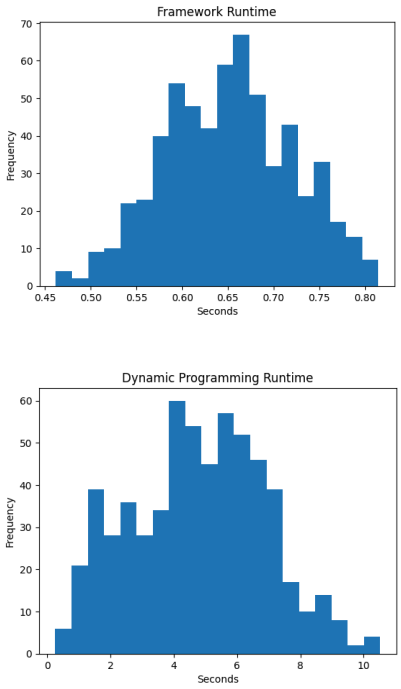

Figure 7: Framework Runtime vs. DP Runtime

FUTURE ADDITIONS

There are several opportunities for the further expansion of the fault recovery framework. Support for more complex inter-task interactions can be implemented. This may include handling intricate task dependencies that form a directed acyclic graph, and injecting new tasks, such as rotating the CubeSat to increase solar panel energy generation as needed, to improve mission outcomes. A greater spectrum of faults could also be supported, including faults which are temporary and repairable, such as the loss of data that must then be recomputed, and faults which increase the time required to complete a certain task, such as decreasing the maximum thrust of onboard thrusters. These improvements would allow the framework to better handle the intricacies of difficult, real-world CubeSat missions and their associated objectives. Additionally, the framework could be implemented on a physical CubeSat system, with functional systems such as attitude control and power management. The framework could be paired with a fault detection system to react to actual onboard failures, while also managing resource utilization to allow for the scheduled mission tasks to execute. Publishing this implementation would allow for CubeSat teams to directly adopt the framework for their own designs and mission plans, supporting the initial objective of creating a reusable and adaptable method for recovering from onboard faults.

To further optimize scheduling within the fault recovery framework, tasks from the initial mission schedule could be intelligently merged and rearranged when doing so improves efficiency, and does not interfere with mission objectives. For example, separated communication tasks can be batched to reduce repeated overhead. This initial optimization would increase the resource and time consumption of the initial mission plan, potentially allowing for a greater subset of mission objectives to be preserved during fault recovery.

CONCLUSION

Onboard failures serve as a significant problem for CubeSat missions, exacerbated by the difficulty of predicting and reacting to the vast variety of possible failures. While current approaches to solutions exist, they are quite human dependent, lack sufficient generalizability, and focus on the short term. The autonomous fault recovery framework described in this paper demonstrates moderate effectiveness for development and deployment within real-world CubeSat systems to solve these issues. It demonstrates

moderate effectiveness in schedule rearrangement across a wide range of simulated mission plans and faults with a 22% median margin of error across 20 simulated trials, and only a 3.54% margin of error for task selection. Further modifications to the model architecture and training process, such as refining the observation space and changing the iteration structure of the DQN environment, may improve the accuracy and increase consistency.

Overall, this framework lays the foundation for the usage of AI systems to enhance the robustness and autonomy of CubeSat operations. By reducing human dependency while exhibiting strong generalizability across diverse scenarios, this approach represents a pathfinder toward more capable autonomous fault management for future missions - especially crucial as CubeSats continue to increase in complexity and ambition. Such technologies enable reliable, cost-effective exploration and utilization of space. Importantly, making these autonomous fault recovery capabilities available as an accessible resource has the potential to democratize access to space missions for teams with limited personnel and budget constraints, allowing them to leverage AI for enhancing the resilience and autonomy of their CubeSat systems and make their ventures into space.

References

1. Swartwout, M, "CubeSats and Mission Success: A Look at the Numbers," 2016 CubeSat Developers' Workshop, Cal Poly, San Luis Obispo, CA, April 2016.

2. Rao, J., Pace, J., Williams, J., Mackey, R., He, L., Menegazzo, C., "A Reusable Framework for Fault Detection and Isolation in Small Satellites," 2023 Small Satellite Conference, Utah State University, Logan, UT, August 2023.

3. Alanazi, A., Straub, J., "Statistical Analysis of CubeSat Mission Failure," 2018 Small Satellite Conference, Utah State University, Logan, UT, August 2018.

4. Liang, J., Zhu, Y., Luo, Y., Zhang, J., Zhu, H., "A precedence-rule-based heuristic for satellite onboard activity planning," *Acta Astronautica* 178, January 2021.

5. Mnih, V., Kavukcuoglu, K., Silver, D., et al, "Human-level control through deep reinforcement learning," *Nature* 518, February 2015.

6. Lobo, J.S., Ghiglino, P., Escobedo, S.L., Rivo, M.S., "Design of a Model-Based Failure Detection Isolation and Recovery System for Cubesats," 8th European Conference for Aeronautics and Aerospace Sciences (EUCASS), July 2019.

7. Asiri, M.Y., "Novel Multipath TCP Scheduling Design for Future IoT Applications," Deakin University, Australia, October 2023.