Take-off Autonomous Resilience System

Benjamin Berton and Alexia Sanon

McGill University

Montreal, Quebec, Canada

alexia.sanon@mail.mcgill.ca
benjaminberton64@gmail.com

Abstract -

This paper presents the development of an autonomous copilot agent designed to assist human pilots during critical take-off phases, particularly in managing abnormal situations such as bird strikes. The system combines an ACT-R cognitive architecture with a random forest classifier to dynamically assess pilot control and provide adaptive automation support. Using flight simulation data from six participants with varying experience levels, the agent evaluates pilot performance across multiple parameters including flight control input stability, runway centerline deviation, and reaction times to critical events. The random forest classifier achieved approximately 75% accuracy in distinguishing between high and low pilot control states, enabling real-time adaptation of automation assistance. When pilots exhibit low control, the agent provides enhanced support by managing secondary tasks such as trim adjustment and system monitoring, while maintaining pilot engagement and situational awareness during high-control periods. Results demonstrate the potential of interpretable machine learning techniques in aviation adaptive automation, though limitations in training data and sequential processing capabilities suggest areas for future improvement. This research contributes to addressing the automation conundrum in aviation by balancing safety enhancement with pilot engagement.

Index Terms – Adaptive automation; Random Forest classifier; cognitive modeling.

I. Introduction

The goal of this project is to develop an autonomous "copilot" agent that collaborates with a human pilot during the critical take-off phase of flight, managing abnormal situations such as those caused by a bird strike. This scenario is simulated in X-Plane 11 using a Very Light Jet (Cessna Citation Mustang) operated by a single pilot. The mission begins on the runway, proceeds through takeoff, and continues until reaching a safe altitude of 1,500 feet Above Ground Level (AGL). During the sequence, a bird strike may occur before or after the decision speed (V1), causing an engine failure in either the left or right engine.

To adaptively support the pilot, the autonomous agent employs a random forest classifier to evaluate the pilot's level of control throughout the event. Based on the classification (High or Low Control), the agent dynamically adjusts its level of automation. When the pilot exhibits low control, the agent provides high support by taking on additional tasks. Conversely, when the pilot demonstrates high control, the agent reduces its involvement to maintain the pilot's engagement and situational awareness (SA).

The overarching goal of this agent is to emulate the essential functions of a human copilot, such as monitoring, annunciating events, and managing systems, while allowing the human pilot to focus on controlling the aircraft. Engine failure after takeoff presents significant cognitive and physical challenges. In such scenarios, the single pilot must retain sufficient control to build and maintain SA. However, if the pilot is already struggling before the bird strike, the agent alleviates cognitive and physical load to ensure a safer and more manageable response.

This project emphasises the agent's ability to accurately classify the pilot's state and adapt its behaviour accordingly, rather than delving deeply into the internal mechanisms of the cognitive model within the simulated environment. The agent is developed within the ACT-R (Adaptive Control of Thought-Rational, [1]) cognitive architecture, which models human-like cognition to enhance real-time decision-making and adaptability.

To achieve real-time classification, we implemented a random forest classifier, a robust ensemble learning algorithm that combines multiple decision trees to enhance prediction accuracy. By leveraging ensemble learning principles, the model produces a more reliable and effective classification of pilot control, ensuring dynamic support tailored to the situation.

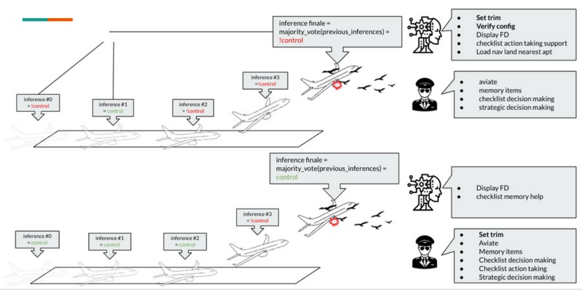

Figure 1: Overview of the agent support based on the inferred pilot control of the aircraft over the takeoff phase

II. Related Work

The aviation industry has long been interested in adaptive automation, driven by human factors and human-computer interaction research. A recurring challenge in this domain is the *automation conundrum*, also known as the *lumberjack effect*. While increasing automation has significantly enhanced safety in the aeronautical sector, it has introduced complexity for pilots interacting with these systems. This complexity often manifests in edge cases or system failures, where pilots may struggle to manage degraded situations due to faulty or incomplete mental models, skill erosion, and the startle effect. Additionally, high levels of automation that exclude pilots from the operational loop degrade their *situation awareness* —a key factor contributing to human error, which remains the primary cause of aviation accidents today.

Adaptive automation seeks to address these issues by dynamically adjusting the level of automation based on the pilot's real-time needs. By being human-aware, adaptive systems monitor pilots' cognitive workload, level of control, and even physiological states using health data, as proposed by Hauptman and McNeese [3]. Such systems act as a safety net, intervening only when necessary, thereby balancing automation benefits with the need to maintain pilot engagement and SA.

Blum et al. [2] explored this concept by combining unsupervised learning with ACT-R cognitive modelling to anticipate pilot performance. Their agent clustered pilot behaviours and used simulated mental models to predict whether a pilot would perceive or miss alarms. Inspired by their hybrid approach, our project replaces their unsupervised learning algorithm with a supervised learning method—a random forest classifier. The goal of our agent is to infer the pilot's level of control in real-time and dynamically adjust its support accordingly, enhancing safety and reducing pilot workload during high-stress scenarios, such as an engine failure after takeoff.

III. Methodology

A. *Agent Design*

1) **Cognitive Architecture**: ACT-R is a widely used and comprehensive cognitive architecture that models cognition as a hybrid of symbolic and sub-symbolic processing. It follows a modular structure to represent cognitive processes, mirroring the interaction of different functional areas of the human brain. These modules enable the modeling of memory processes, as well as basic perceptual and motor capabilities. Communication between modules occurs through the exchange of memory representations, called chunks, which are stored in buffers—each capable of holding one chunk at a time. Based on the state of these buffers, production rules (if-then statements) can be triggered to modify buffer contents.

We used ACT-R to model the behaviour of a human copilot by integrating an appropriate checklist into the model's declarative memory. Productions were defined to enable the agent to work through this checklist, interpreting the aircraft's state (e.g., alarm states, indicated airspeed) through a virtual cockpit to which the model has access. The agent then issues corresponding actions based on the checklist items. The virtual cockpit includes a parameter that is not visible to the human participant: the inferred pilot control class. This class, updated every 10 knots to reflect the majority vote of the previous 10-knot window, is used by the agent to adapt its behaviour. Upon detecting an alarm, the agent references this class to adjust its level of support.

The primary difference in this implementation of the agent's behaviour lies in its response when the pilot is classified as having poor control of the aircraft. In addition to displaying the flight director (a visual target on the Primary Flight Display indicating the required attitude for a safe and efficient climb), the agent compensates for the asymmetrical thrust caused by engine failure by adjusting the trim in the opposite direction of the failed engine. This action is something the pilot would typically do, but in this high-stress scenario, they are likely overwhelmed by the need to maintain flight symmetry—balancing the rudder pedals while simultaneously controlling roll and pitch with the yoke. Moreover, the rudder trim control is not easily accessible and requires several incremental inputs (approximately 50 clicks over 10 seconds). To ensure the human pilot remains aware of the agent's actions, the agent announces each step aurally.

2) *Machine Learning Component*: The machine learning algorithm we opted for was Random Forest. A decision tree is a type of predictive model used in machine learning and statistics. It is a non-parametric machine learning method, meaning that it does not assume a specific functional form (e.g., linear or polynomial) for the relationship between inputs and outputs. Decision trees are popular for classification and regression tasks due to their interpretability and simplicity. They involve the greedy selection of the best split point from the dataset at each step. Bagging ensembles are an approach to reduce this variance and thereby increase model performance. In this algorithm, multiple weak learner models (i.e. decision trees) produce predictions on a series of bootstrap samples. A random forest is a versatile ensemble machine learning algorithm primarily used for classification and regression tasks. It works by constructing multiple decision trees during training and combining their outputs (via averaging for regression or majority voting for classification) to produce a more robust and

accurate model. One key advantage of using a random forest for classification over algorithms like logistic regression, support vector machines (SVM), and k-nearest neighbors (KNN) is its ability to handle complex, non-linear relationships in the data without requiring extensive pre-processing or feature engineering. Unlike logistic regression, which assumes a linear relationship between input features and the output, random forests can model highly non-linear decision boundaries. Compared to SVM, random forests are often more scalable to large datasets and do not require tuning complex hyperparameters like kernels. Additionally, unlike KNN, which can be computationally expensive at prediction time and is sensitive to the choice of distance metrics, random forests provide faster predictions after training and are robust to noise and irrelevant features due to their feature selection mechanism at each tree split.

Following [1] and [2], we implemented a custom random forest. To decide on the best "split point", i.e. the best feature to split on, we chose to compute the weighted Gini Index (or Gini Impurity) of the splits. It measures the impurity reduction of a dataset or subset of data based on how well the data points are grouped by their classes. The weighted Gini index is an extension of the standard Gini index that accounts for the size of each subset when measuring the overall impurity after a split. This ensures that larger subsets have a proportionally greater influence on the Gini impurity score than smaller subsets. For a dataset S split into k subsets $\{S_1, S_2, \ldots, S_k\}$, the weighted Gini index is calculated as

$$Gini_{\text{split}} = \sum_{j=1}^{k} \frac{|S_j|}{|S|} \left(1 - \sum_{i=1}^{C} p_{i,j}^2\right) \tag{1}$$

Where:

- *S: The entire (sub) dataset being split*
- *Sj: The* j-th subset of the split.
- |Sj|: The size of subset Sj.
- |S|: The total size of the original dataset S.
- C: The number of classes.
- Pi,j: The proportion of data points in class i, within subset Sj.

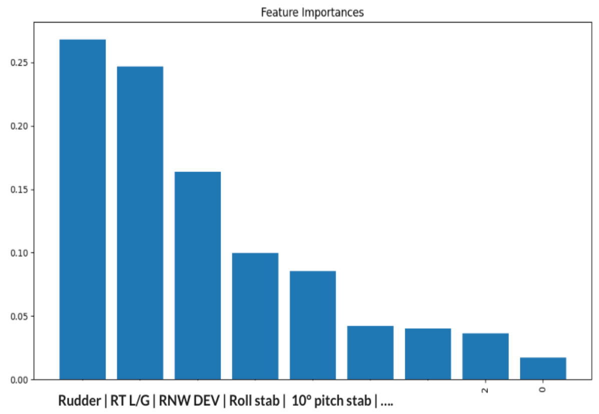

Figure 2: Gini Index for each feature

The dataset consisted of flight logs obtained from simulated takeoff tasks performed by n=6 participants. The participants included three "novices" with no prior experience in flight simulators and three "experts" with over 100 hours of cumulative experience in flight simulation. The selected features were:

- *Flight Control Input Stability*: Measures the variance in control inputs (pitch, yaw, roll), during take-off. We measured it using RMS error of control inputs over a 10kts window.
- *Runway Centerline Deviation*: This is the distance of the aircraft's lateral position from the runway centerline. We measured it using RMS error of centerline deviation over a 10kts window.
- *Pitch Angle Consistency*: Stability of the target pitch of 10° during initial climb (after rotation at V1=VR=90kts). We measured it using RMS error from 10° pitch angle.
- *Reaction Time to VRotate*: This is the time taken from the rotate callout to initiating the rotation maneuver. We measured it by tracking the milliseconds from the event timestamp to having a pitch > 1°.
- *Reaction Time to Retract Flaps*: This is the time taken to retract flaps upon reaching 110 knots indicated airspeed (KIAS). We measured it by tracking the milliseconds from 110 KIAS crossing to flap retraction input.
- *Reaction Time to Retract Landing Gear*: This represents the time taken to retract the landing gear after achieving a positive vertical speed. We measured it by tracking the milliseconds from the positive rate of climb to gear retraction input.

These features were extracted and logged to a .csv files. It is important to note that we considered discarding the first 30kts of flight from the logged flight data, as those were identified as redundant data since the first 30kts of flight are more or less the same regardless of the pilot. However, it became quickly evident the model's performance was not affected by them, so we did not change anything in the end.

Our implementation of Random Forest's classification accuracy depends on maximum depth, number of trees, and minimum sample split. The table and figures below show the effect of these parameters on the algorithm's performance. Note that we kept the sample size constant as we were more focused on getting the other parameters right, since the sample size needs to stay some value under 1 to ensure tree diversity and good generalization, we chose an arbitrary value and stuck with it.

Table 1: Effects of parameters on model performance

Parameter	Effect on Model Performance	Typical Impact
Sample size	Controls how much of the dataset each tree sees.	Smaller sample size increases tree diversity, better generalization but might reduce accuracy per tree.
Number of trees	Number of trees in the forest.	More trees improve performance but increase computation; fewer trees increase variance and decrease accuracy.
Maximum depth	Limits the depth of each tree.	Shallow trees prevent overfitting but may underfit; deeper trees may overfit and increase complexity.
Minimum sample split	Determines the minimum number of samples required for a split.	Small value causes overfitting; large value avoids overfitting but may underfit.

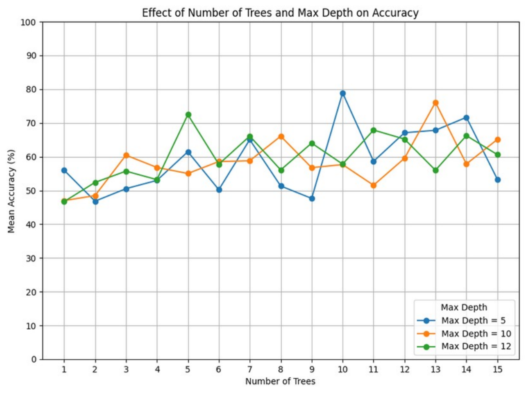

Figure 3: Effect of Maximum Depth on performance

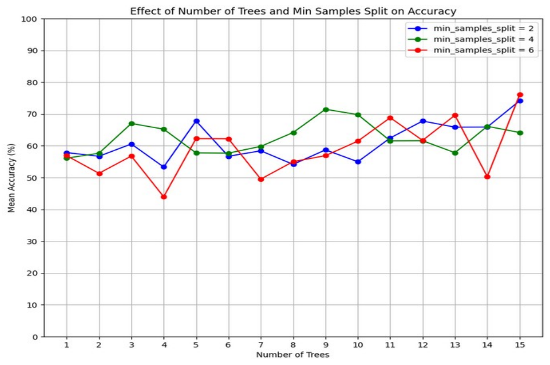

Figure 4: Effect of Minimum Sample Split on Performance

As can be seen in the figures above, a maximum depth of 12 offered more stable/consistent mean accuracy scores across different tree numbers. This is likely due to the fact that values under 10 might be underfitting. The algorithm is most unstable with value 6, likely due to underfitting as the value might be too large when considering the size of the dataset (only about 129 entries). By observing both plots, we identified four possible values for the size of our forest: 10, 12, 13 and 15. These values are the ones where either minimum sample split or maximum depth scored particularly high. The first step was then to settle on a value for the number of trees, from there other plots were produced while testing the possible values for the other two parameters.

After testing different values for each parameter and carefully observing their effects on the algorithm's performance, we settled on the following values:

Table 2: Parameter Values

Parameter	Value
K	4
Sample Size	0.65
Minimum Sample Split	5
Maximum Depth	12
Number of Trees	15

IV. Results

For performance evaluation, we implemented a k-Fold Cross Validation. It is a statistical technique used to assess the performance of a machine learning model by systematically splitting the data into multiple subsets (or folds) to train and test the model. Here is how it works:

- k is a user-defined parameter (4 in our case).
- The dataset is split into k equally sized (or nearly equal) subsets or folds.
- For each of the k iterations:
    - Use k−1 folds for training.
    - Use the remaining fold for testing (referred to as the validation fold).
- Repeat the process k times, with each fold being used as the validation set exactly once.

The value for k was chosen after testing different values as can be seen in Fig.4.

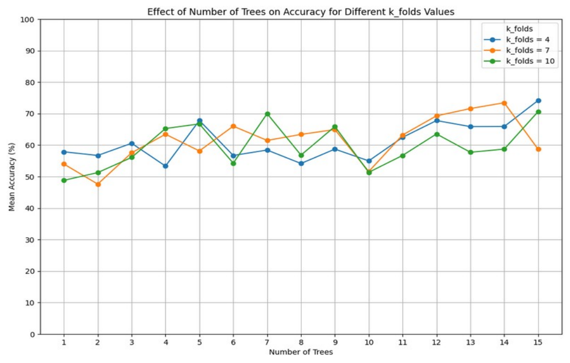

Figure 5: Effect of k on Performance

A smaller number of folds means each fold has more training data.

Next in the evaluation, we computed the performance metric (prediction accuracy here) for each iteration eq (2); Take the average of these metrics to get an overall estimate of the model's performance eq (3):

$$Accuracy = \frac{Total\,Correct\,Predictions}{Total\,Tue\,Values} \cdot 100$$

(2)

$$Mean\,Accuracy = \frac{Accuracy\,Scores}{k}$$

(3)

With the specified values for our parameters, we ran the algorithm with different seeds. As can be seen in Fig.5, the seed that yielded the highest mean accuracy score (~75%) was 42. We also ran an instance of Scikit-Learn, which performed better than our implementation by roughly 13%.

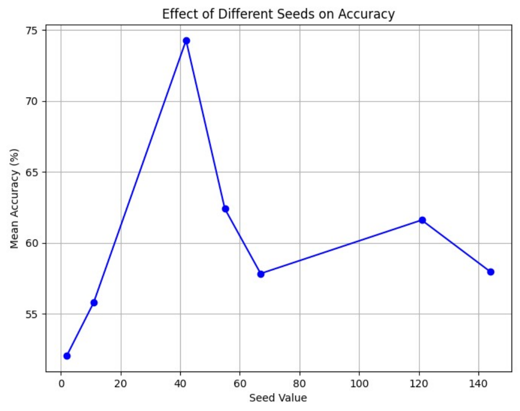

Fig.6 Seeds' Effect on Performance

To achieve real time classification, the model is first trained on the available dataset, exported, then later loaded and queried when needed during a flight (i.e. every 10 kts).

The figure below displays outputs from running the agent in real time with ACT-R and the simulation. Up to a certain point the agent makes its predictions, then decides on one classification by way of majority vote. As can be seen, based on the pilot's perceived "expertise", the agent's degree intervention after a bird strike is different: the agent takes on more tasks when dealing with a "Novice" pilot (i.e. a pilot that is not completely in control) to ease the mental and physical load on the pilot.

The following figure illustrate the implementation of the agent and the connection between the Random Forest classifier and the ACT-R model.

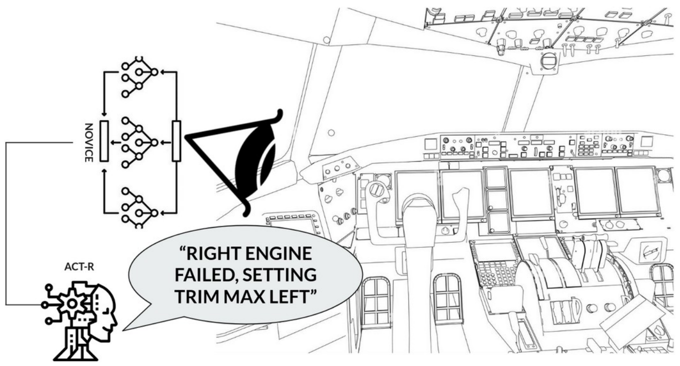

Figure 6: Synoptic of the agent observing the pilot's behaviour in real-time and providing the ACT-R model with the inferred state of control

V. Conclusion

We chose a random forest classifier for this project due to its ability to handle heterogeneous data, its ease of implementation, and its low computational requirements for both training and real-time inference. Additionally, random forests offer greater interpretability compared to deep learning models, which are often difficult to explain. In aviation, a safety-critical domain with strict certification processes, the explainability of algorithms is essential to ensure trust and accountability.

Our random forest classifier was implemented using a sliding window approach, capturing snapshots every 10 milliseconds for each 10-knot increment of airspeed. These snapshots were analysed to classify whether each segment of the takeoff phase was under control or not. This same approach was applied for real-time inference. While a simpler threshold-based system could have been used—flagging poor control whenever the aircraft exceeded its safety envelope—we opted for an AI-based solution to account for nuances in pilot behaviour. For example, an experienced pilot performing an avoiding maneuver to evade a bird strike might temporarily exceed the safety envelope, but this should not be interpreted as poor control. A threshold-based system could incorrectly classify such behaviour, whereas the random forest classifier considers the broader context.

However, there is a limitation in our approach: random forests are not inherently designed to handle sequential data, such as the time-series nature of takeoff events. While we considered using a recurrent neural network (RNN), which is better suited for time-based data, we chose random forests to maintain model interpretability. To address this limitation, we incorporated speed as an input feature and used majority voting for the final classification. This allowed the agent to evaluate the entire takeoff phase holistically rather than focusing solely on the current state of the aircraft.

Due to the lack of access to real-world pilot data, we recruited participants with varying levels of experience in flight simulation. While this provided valuable data for training and testing, it is important to note that the "experienced" participants were not professional pilots. In future work, such a system should be trained on data from professional pilots, with classifications validated by a certified flight examiner. Our current approach relied on the assumption that participants with no flight simulator experience would naturally exhibit poor control during takeoff. While this provided a starting point, it is not a robust scientific method and highlights the need for more rigorous data collection and evaluation procedures.

In conclusion, while our implementation has limitations, it demonstrates the potential of using interpretable machine learning techniques, such as random forests, to enhance adaptive automation in aviation. Future developments should focus on incorporating professional-grade training data, exploring more advanced sequential models, and refining the classification methodology to improve reliability and generalisation.

References

[1] Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York, NY: Oxford University Press. ISBN 0-19-532425-0.

[2] Blum, S., Klaproth, O., & Russwinkel, N. (2022). *Cognitive Modeling of Anticipation: Unsupervised Learning and Symbolic Modeling of Pilots' Mental Representations*. Topics in Cognitive Science. https://doi.org/10.1111/tops.12594

[3] Hauptman, A. I., & McNeese, N. J. (2022). *Overcoming the Lumberjack Effect Through Adaptive Autonomy*. Proceedings of the Human Factors and Ergonomics Society Annual Meeting, 66(1), 1075-1079. https://doi.org/10.1177/1071181322661372

[4] "How to Implement Random Forest From Scratch in Python," Machine Learning Mastery, Nov. 13, 2016. https://machinelearningmastery.com/implement-random-forest-scratch-python/.

[5] A. Shafi, "Sklearn Random Forest Classifiers in Python Tutorial," www.datacamp.com, Feb 23. https://www.datacamp.com/tutorial/random-forests-classifier-python