Université de Montréal

# Detecting Pre-Error States and Process Deviations Resulting from Cognitive Overload in Aircraft Pilots

par
Massimo Pietracupa

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en computer science

December, 2023

Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé:

**Detecting Pre-Error States and Process Deviations Resulting from Cognitive Overload in Aircraft Pilots**

présenté par:

Massimo Pietracupa

a été évalué par un jury composé des personnes suivantes:

| | |
|---|---|
| Maxime Mignotte, | président-rapporteur |
| Claude Frasson, | directeur de recherche |
| Utsav Sadana, | membre du jury |

Mémoire accepté le: . . . . . .

## RÉSUMÉ

Les pilotes d'avion sont constamment confrontés à des situations où ils doivent traiter des quantités importantes de données en très peu de temps, ce qui peut conduire à des erreurs. Nous avons créé un système de détection des écarts capable d'auditer le cockpit en temps réel pour détecter les actions qui ont été incorrectement ajoutées, omises ou qui n'ont pas été effectuées dans le bon ordre. Ce modèle évalue les écarts en se basant sur les données hiérarchiques des tâches trouvées dans le modèle de référence ontologique pour les procédures de pilotage, qui contient des procédures de référence basées sur la connaissance et rassemblées par des experts dans le domaine. Les actions des pilotes sont comparées aux séquences de référence de l'ontologie à l'aide de l'algorithme Needleman-Wunsch pour l'alignement global, ainsi que d'un réseau LSTM siamois. Une API pouvant être étendue à plusieurs simulateurs aérospatiaux, ainsi qu'un Runner, ont été créés pour permettre au Deviation Framework de se connecter au simulateur XPlane afin de surveiller le système en temps réel. Des données créées synthétiquement et contenant des mutations de séquences ont été analysées à des fins de test. Les résultats montrent que ce cadre est capable de détecter les erreurs ajoutées, omises et hors séquence. En outre, les capacités des réseaux siamois sont exploitées pour comprendre la relation de certaines anomalies de la chaîne de séquence afin qu'elles puissent être correctement ignorées (comme certaines tâches qui peuvent être exécutées dans le désordre par rapport à la séquence de référence). Les environnements de simulation enregistrant les données à une fréquence de 10 Hz, une valeur de 0.1 seconde constitue notre référence en temps réel. Ces évaluations de déviation peuvent être exécutées plus rapidement que notre contrainte de 0,1 seconde et ont été réalisées en 0,0179 seconde pour une séquence de décollage contenant 23 actions, ce qui est nettement plus performant que les modèles suivants de l'état de l'art. Les résultats de l'évaluation suggèrent que l'approche proposée pourrait être appliquée dans le domaine de l'aviation pour aider à détecter les erreurs avant qu'elles ne causent des dommages.

En outre, nous avons formé un modèle d'apprentissage automatique pour reconnaître les signaux de pré-erreur dans le cortex cingulaire antérieur (CCA) à l'aide des données

de test Flanker de l'ensemble de données COG-BCI, qui peuvent ensuite être utilisées pour détecter les états de pré-erreur chez les pilotes d'avion. Divers modèles d'apprentissage automatique ont été appliqués à l'ensemble de données, notamment des machines à vecteurs de support (SVM), des forêts aléatoires, un double modèle de réseau neuronal convolutif (CNN) et un modèle Transformer. Au-delà des conclusions typiques de l'étude, notre objectif s'étend à l'évaluation de l'applicabilité du modèle dans un domaine secondaire, à savoir l'évaluation du pouvoir discriminant des classificateurs pendant les procédures de décollage pour les pilotes d'avion. Les résultats de l'analyse de l'ensemble de données FLANKER ont révélé la supériorité du modèle transformateur, avec des réductions notables des faux négatifs et un score final macro moyen F1 de 0,610, et un score final macro moyen F1 de 0,578 sur les données pilotes. Comme nous prévoyons une augmentation des performances du classificateur avec davantage de données d'entraînement et des bandes d'interrogation étendues, cette étude jette les bases d'une recherche plus poussée sur la prédiction des états erronés et les modèles d'optimisation de l'apprentissage automatique pour les ICB et les applications du monde réel.

**mots clés: Réalité virtuelle, simulation, contrôle des tâches, gestion des erreurs, réseau neuronal, EEG, apprentissage automatique, transformateurs, cortex cingulaire, prédiction de l'état d'erreur**

## ABSTRACT

Aircraft pilots are constantly undergoing situations where they must process significant amounts of data in very small periods of time, which may lead to mistakes. We have created a deviation detection system that is capable of auditing the cockpit in real time to detect actions that have been incorrectly added, omitted, or done out of sequence. This model assesses deviations based on hierarchical task data found in the Ontological Reference Model for Piloting Procedures, which contains knowledge-based reference procedures assembled by experts in the domain. Pilot actions are compared to ontology reference sequences using the Needleman-Wunsch algorithm for global alignment, as well as a Siamese LSTM network. An API that can be expanded to several Aerospace Simulators, as well as a Runner, was created to enable the Deviation Framework to connect to the XPlane simulator for real-time system monitoring. Synthetically created data containing sequence mutations were analyzed for testing. The results show that this framework is capable of detecting added, omitted, and out of sequence errors. Furthermore, the capabilities of Siamese networks are leveraged to understand the relation of certain sequence chain anomalies so that they can correctly be ignored (such as certain tasks that can be performed out of order from the reference sequence). With simulation environments recording data at a frequency of 10Hz, a value of 0.1 seconds is our real-time benchmark. These deviation assessments are capable of being run faster than our 0.1 second requirement and have been clocked at 0.0179 seconds for one Takeoff sequence containing 23 actions - significantly outperforming the next state of the art models. The evaluation results suggest that the proposed approach could be applied in aviation settings to help catch errors before harm is done.

Moreover, we have trained a machine learning model to recognize pre-error signals in the anterior cingulate cortex (ACC) using Flanker test data from the COG-BCI dataset, which can be subsequently employed to detect pre-error states in aviation pilots. Various machine learning models were applied to the dataset, including Support Vector Machines (SVM), Random Forests, double Convolutional Neural Network (CNN) model, and a Transformer model. Moving beyond typical study conclusions, our objective extends to

assessing model applicability in a secondary domain —evaluating the classifiers' discriminative power during takeoff procedures for aviation pilots. Results from the analysis of the FLANKER dataset revealed the superiority of the transformer model, with notable reductions in false negatives and a final macro averaged F1 score of 0.610, and a final macro averaged F1 of 0.578 on the Pilot data. As we anticipate increases in classifier performance with more training data and extended polling bands, this study lays the groundwork for further research in erroneous state prediction and machine learning optimization models for BCI and real-world applications.

**Keywords: Virtual Reality, Simulation, Task control, Error Management, Neural Network, EEG, Machine Learning, Transformers, Cingulate Cortex, Error-State Prediction**

## CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

## LIST OF ABBREVIATIONS

| | |
|---|---|
| ACC | Anterior Cingulate Cortex |
| BCI | Brain-Computer Interfaces |
| COG | Cognitive |
| CNN | Convolutional Neural Network |
| EEG | Electroencephalogram |
| EF | Engine Failure |
| FDM | Flight Data Monitoring |
| FIR | Finite Impulse Response |
| ICA | Independent Component Analysis |
| LOSA | Line Operations Safety Audit |
| LSTM | Long-Short Term Memory |
| RBF | Radial Basis Function |
| SMOTE | Synthetic Minority Oversampling Technique |
| SVM | Support Vector Machine |
| UDP | User Datagram Protocol |

To my Grandmother, Giovanna Farinaccio

## ACKNOWLEDGMENTS

In the culmination of this academic journey, I am filled with profound gratitude for the countless individuals and experiences that have shaped the trajectory of this thesis.

First and foremost, I extend my heartfelt appreciation to my supervisor, Dr. Claude Frasson, and Dr. Hamdi Ben Abdessalem, whose unwavering guidance, expertise, and encouragement have been indispensable throughout every phase of this research. Your mentorship has not only sharpened my academic skills but has also inspired me to strive for excellence in all endeavors. We also gratefully acknowledge the generous support from NSERC-Alliance, CRIAQ, CAE, Bombardier, and BMU, whose funding made this research possible. Without their invaluable contributions, this endeavor would not have been possible.

I am indebted to my family (Maria D'Annessa, Pat Pietracupa, Bianca Pietracupa), my aunts, uncles, and cousins (in no particular order: Adamo D'Annessa, Kyara D'Annessa, Emilio D'Annessa, Brenda Bucci, Jonathan D'Annessa, James D'Annessa, Joseph D'Annessa, Jack D'Annessa, Tizianna Bibbo, Claudia Feocheri, Konnie Feocherri, Micheline Pietracupa) for their boundless love, encouragement, and patience throughout this journey. Your unwavering belief in my abilities has been a constant source of strength and motivation. The most important thing in the world is family, and I consider myself blessed every single day.

Special thanks are extended to Samantha Cavaliere, whose unwavering presence, love, and support have been my rock through every challenge and triumph. Your encouragement and belief in me have fueled my determination and inspired me to push beyond my limits.

I also extend my heartfelt appreciation to Samantha's family (Renaldo Cavaliere, Caterina De Luca, Stefano Cavaliere, Marie Paolonie, Esterina Molinaro) for their warm acceptance and embrace. Your kindness and hospitality have made me feel like a cherished member of the family, and for that, I am truly grateful.

To my friends, colleagues, Uber squad, Soccer squad, and the "brudders", who have provided encouragement, understanding, and a shoulder to lean on along the way, I

extend my sincerest gratitude and respect. Your support has illuminated even the darkest of moments and made this journey all the more meaningful.

[Italian] Ai miei nonni, Giuseppe D'Annessa, Onorina [cognome] per il loro incondizionato sostegno e amore. La vostra fiducia in me è stata una costante fonte di forza e ispirazione, e il mio aiuto sarà sempre a vostra disposizione quando ne avrete bisogno.

I dedicate this achievement to my beloved grandmother, Giovanna Farinaccio, who was not only a source of unwavering love and support but also a guiding light in my life. Though you are no longer with us to witness the culmination of this journey, your spirit continues to inspire me every day. Your memory will forever be cherished, and this project stands as a tribute to your enduring legacy. Tu sei il mio fiorellina, Ti voglio molto bene nonna e mi manchi profondamente.

Finally, I extend this dedication to anyone who has ever dared to dream, to question, and to pursue knowledge with unwavering determination. May this thesis serve as a testament to the power of perseverance, curiosity, and the relentless pursuit of excellence.

# CHAPTER 1

# INTRODUCTION

## 1.1 Context

In recent years, the aviation industry has experienced unprecedented growth, with air travel becoming increasingly accessible and integral to global connectivity. As the demand for air transportation continues to soar, the imperative for ensuring uncompromising safety standards has never been more pronounced. While technological advancements have ushered in an era of unprecedented safety improvements, human factors remain a critical determinant of aviation safety.

Aircraft pilots, entrusted with the responsibility of safely operating complex machines through dynamic and often unpredictable environments, stand at the forefront of ensuring the safety of air travel. Despite their extensive training and expertise, pilots are susceptible to errors that can have profound consequences for flight safety. From minor deviations in procedures to critical lapses in judgment, human errors constitute a persistent challenge in the quest for aviation safety.

In response to this challenge, the aviation industry has continuously sought to develop and implement innovative solutions aimed at mitigating the risks associated with human error. Central to this endeavor is the integration of advanced technologies that augment pilot capabilities and enhance operational safety. From flight management systems to advanced cockpit displays, technological innovations have played a pivotal role in advancing aviation safety by providing pilots with invaluable support and decision-making aids.

However, while technological solutions have undoubtedly contributed to improvements in aviation safety, the quest for enhancing human performance and error mitigation remains an ongoing pursuit. Recognizing the need for proactive measures to address the complex interplay between human factors and technology, this research strives to develop a novel approach to enhance aviation safety.

## 1.2 Problem

The aviation industry, while marked by remarkable advancements in technology and training methodologies, continues to grapple with the inherent fallibility of human operators [1–3]. Aircraft pilots are constantly undergoing situations in which they must process significant amounts of data in very small periods of time. Despite stringent safety protocols and procedures, errors made by pilots still pose significant risks to flight safety. The ability to detect whether tasks are being performed based on a set of guidelines can allow pilots the foresight to make adjustments earlier in order to prevent failures, or even to increase their respective efficiency. An observational methodology study using a Line Operations Safety Audit (LOSA) (where expert observers are placed in the cockpit during normal flights to record threats) was performed over a period of 15 years with confidential data collected on more than 3500 domestic and international flights. Supported by the Federal Aviation Administration and the International Civil Aviation Organization, their reports indicate that some of the most common types of errors by Aircraft Pilots include a conscious failure to adhere to procedures or regulations (such as performing checklists from memory) and following procedures with wrong executions (incorrectly entering data into the flight management computer) [4]: These are issues that can be significantly reduced with the help of a real-time deviation detection systems or predictive error systems, which would give pilots the extra time needed to perform necessary corrections, though these systems are challenging to create for complex systems such as an aircraft. As a result, the following hypothesis are formulated for this paper:

- **Hypothesis: 1**: Is it possible to create a **deviation model** to determine problematic Pilot actions that are added, omitted, and out of sequence errors during flight that is capable of understanding the underlying relations of tasks?
- **Hypothesis: 2**: Will this deviation detection model architecture be capable of executing in real-time (real time defined as simulation frequency of 10Hz)?
- **Hypothesis: 3**: Is it possible to train a machine learning model that can identify pre-error signals, which can serve as indicators of pre-error states in aviation

pilots?

## 1.3 Contribution

In addressing these problems, this project seeks to develop tools and a comprehensive deviation detection system aimed at intercepting pilot errors in real-time. By leveraging dynamic programming algorithms, and ontological aircraft references, this system will be capable of identifying deviations from established protocols or best practices during flight operations. Moreover, the project aims to devise a predictive model that can anticipate states wherein pilots are more prone to committing errors, through EEG data gathered from real pilots in a simulation environment. This predictive capability will enable future technologies to preemptively intervene, mitigating potential risks and bolstering overall flight safety. As a result, the objectives present in this work are as follows:

- **Objective 1**: An XPlane API that is easy to incorporate in python and allows for several applications to communicate with each other, and Xplane in real-time.
- **Objective 2**: A deviation detection model that can determine problematic Pilot actions that are added, omitted, and out of sequence during flight that is also capable of understanding the underlying relations of tasks. This model can also be run in real-time, while significantly outperforming speeds of other state-of-the-art models.
- **Objective 3**: A machine learning model that can identify ACC pre-error signals in Aircraft Pilots, which can detect pre-error states.

## 1.4 Document Structure

This project is structured to provide a comprehensive exploration of the development and implementation of innovative technological solutions aimed at enhancing aviation safety through the mitigation of pilot errors. The following sections delineate the organization and content of this document:

Chapter 2, Background and State of the Art: This section provides the background necessary for concepts found in this document. It also provides a comprehensive survey

of existing research and developments related to aviation safety, automatic process deviation, and BCI devices. It explores the current state of knowledge, identifies gaps in research, and establishes the theoretical foundation for the subsequent research efforts.

Chapter 3, Methodology: In this section, the methodology employed in the research is detailed, encompassing the design and implementation of the deviation detection system and predictive model. This includes a description of data collection methods, experimentation, analytical techniques, and model validation procedures.

Chapter 4, Results and Discussion: This section provides a critical analysis and discussion of the findings, implications, and contributions of the research.

Chapter 5, Conclusion: The conclusion section summarizes the key findings, contributions, and implications of the research, reaffirming its significance in advancing aviation safety. It also offers concluding remarks and recommendations for future research and practical implementation.

# CHAPTER 2

# BACKGROUND AND STATE OF THE ART

In this chapter, we present the necessary literature to understand the work presented in this reading. We first present a brief overview of process deviation detection along with current techniques in aircraft flight data monitoring. We then present background related to machine learning and some of the state-of-the-art technology currently used in aviation for process deviation. We then examine a background in brain-controller interfaces and take a look at the state of the art in pre-error state prediction.

## 2.1 Process Mining - Deviation Detection

We can observe that deviation detection methods in process mining can be categorized as data-driven or knowledge-driven. As their names suggest, these can rely on developing process models from acquired data, or by having them created by domain experts. For data-driven methods, an average workflow is discovered and then used for comparison with individual activities. However, the average workflow is limited to a strictly sequential view and does not account for concurrent activities or repetitive behaviors [5]. On the other hand, rule-based models are designed for processes that are loosely structured, which limits their applicability for structured Aircraft Processes.

### 2.1.1 Aircraft Flight Data Monitoring (FDM)

In aviation, Flight Data Monitoring (FDM) is a crucial method for ensuring safety and improving operational efficiency. FDM involves the collection, storage, and analysis of flight data from aircraft to monitor and evaluate the performance of various flight parameters, systems, and procedures. This data includes information such as aircraft speed, altitude, heading, engine performance, control inputs, and other relevant variables recorded during flight [6].

One of the key objectives of FDM is to identify deviations from established norms or

expected behavior. These deviations could indicate potential safety issues, operational inefficiencies, or opportunities for improvement. To accomplish this, FDM systems utilize various approaches, one of which involves creating rule-based models based on expert analysis of flight data. The following steps are performed in the FDM process:

**Data Collection**: Flight data is collected from onboard flight data recorders (FDR) or quick access recorders (QAR), which continuously record various flight parameters during the entire duration of a flight. This data is then stored for later analysis [7].

**Data Analysis**: Aviation experts, including pilots, engineers, and safety analysts, analyze the collected flight data to identify patterns, trends, and anomalies. They use their expertise to interpret the data and determine whether observed deviations are normal variations or potential issues that require further investigation [8].

**Rule-Based Model Development**: Based on the analysis of flight data, experts develop rule-based models that codify the relationships between different flight parameters and potential safety or operational concerns. These rules are often derived from industry regulations, best practices, operational procedures, and past incident data [9].

**Rule-Based Monitoring**: The rule-based models are then implemented in FDM systems to continuously monitor incoming flight data in real-time. When the observed flight parameters deviate from the established rules, the FDM system generates alerts or notifications to notify operators or maintenance personnel of potential issues [9].

**Feedback and Iteration**: The effectiveness of the rule-based models is continuously evaluated based on feedback from operational experience, incident reports, and additional data analysis [10]. The models may be refined, updated, or expanded over time to improve their accuracy and relevance.

It should be noted that when navigating Flight Data Monitoring (FDM) systems, certain pitfalls warrant caution. While rule-based models offer significant benefits in enhancing aviation safety and operational efficiency, over-reliance on them may obscure subtle or emerging patterns that don't fit predefined rules [11]. Moreover, developing and maintaining comprehensive rule-based models demands significant expertise and resources, as updating them to reflect changes in operational procedures or regulations can be challenging [11]. Balancing the sensitivity and specificity of rule design is crucial

to minimizing false positive alerts, which can trigger unnecessary interventions, or false negatives, which may overlook genuine safety hazards. Human bias and interpretation also pose risks, as subjective judgments may influence rule design, potentially leading to inconsistencies.

As the number of components, subsystems, and processes within a system grows, so does the number of potential interactions and dependencies between them. Each interaction may require its own set of rules, leading to a combinatorial explosion in the number of rules needed to govern the entire system [11]. Furthermore, managing the complexity of rule sets, ensuring consistency, and resolving conflicts or redundancies become increasingly challenging as systems grow in size and complexity. By learning from data, machine learning models have been implemented and proven that they can adapt to dynamic environments, generalize across diverse scenarios, and discover nuanced interactions that may elude rule-based approaches [11]. This ability to uncover insights from data and make informed decisions autonomously positions machine learning as a transformative approach for managing the complexity of large-scale systems effectively.

## 2.2 Machine Learning

Machine learning is a branch of artificial intelligence (AI) that enables systems to automatically learn and improve from experience without being explicitly programmed [12]. It involves the development of algorithms and models that can analyze data, identify patterns, and make predictions or decisions based on those patterns [12, 13]. Machine learning is becoming increasingly popular because of its ability to solve complex problems across various domains, including finance [14, 15], healthcare [16–18], marketing [19, 20], transportation [21, 22], aviation [23, 24], and more.

One of the key reasons for the popularity of machine learning is its capability to handle large and complex datasets that are beyond the capacity of traditional statistical techniques or rule-based systems [25]. By leveraging advanced algorithms and computational power, machine learning can uncover valuable insights and patterns hidden

within vast amounts of data, enabling organizations to make data-driven decisions and gain competitive advantage [26].

In this section, we will be providing an overview of the machine learning landscape, while introducing certain models that will be mentioned in this paper, in order to give the reader a certain foundational understanding.

### 2.2.1 Machine Learning Landscape

Machine learning, encompasses a diverse array of algorithms and methodologies, which can be broadly categorized into several key areas. Supervised learning, a fundamental pillar, involves training models on labeled data, facilitating predictions on unseen data by learning the mapping between input features and output labels [27]. Unsupervised learning, in contrast, delves into unlabeled data, seeking to unearth hidden patterns, structures, or relationships without the aid of predefined labels [28]. Semi-supervised learning bridges the gap between the two, leveraging a combination of labeled and unlabeled data to enhance model performance, especially in scenarios where labeled data is scarce [29]. Reinforcement learning introduces the concept of agents learning through interaction with an environment, striving to maximize cumulative rewards by making sequential decisions [30]. Deep learning, a subset gaining significant traction, focuses on neural networks with multiple layers, capable of automatically learning hierarchical representations of data, revolutionizing fields such as computer vision and natural language processing [31]. Within this landscape, various specialized areas and techniques, including transfer learning [32], generative adversarial networks (GANs) [33], and ensemble methods, offer tailored solutions to diverse problem domains [34].

#### 2.2.1.1 Support Vector Machines (SVM)

Support Vector Machines (SVM) represent a cornerstone of supervised learning within the landscape of machine learning. SVMs excel in classification and regression tasks by identifying the optimal hyperplane that separates data points of different classes in a high-dimensional space [35]. Positioned within the realm of traditional machine learn-

ing methods, SVMs rely on explicit feature engineering and are particularly effective in scenarios with relatively small sample sizes and high-dimensional data [36]. Their versatility extends to both linearly separable and non-linearly separable data, thanks to the utilization of kernel functions that enable mapping into higher-dimensional spaces. SVMs are renowned for their robustness against overfitting and their ability to capture intricate patterns within the data [37]. As a well-established and interpretable model, SVMs offer valuable insights into decision boundaries and feature importance, making them applicable across a wide range of domains, including bioinformatics, image recognition, and financial forecasting. Despite the emergence of more complex deep learning architectures, SVMs remain a fundamental tool in the machine learning toolbox, valued for their reliability, interpretability, and versatility.

#### 2.2.1.2 Random Forests

Random Forests, a prominent ensemble learning method, occupy a pivotal position within the landscape of machine learning. Positioned alongside traditional machine learning techniques, Random Forests offer a versatile solution for both classification and regression tasks [38]. Comprising an ensemble of decision trees, each trained on a random subset of the training data and features, Random Forests harness the power of collective decision-making to enhance model robustness and generalization [39]. This approach mitigates the risk of overfitting often associated with individual decision trees, rendering Random Forests well-suited for handling complex data with diverse feature types. Their ability to provide feature importance scores facilitates interpretability and insights into the underlying data relationships [40]. As a result, Random Forests find applications across various domains, including finance [41], healthcare [42], and bioinformatics [43], where robust and interpretable models are essential. In the ever-evolving landscape of machine learning, Random Forests stand as a reliable and versatile tool, offering an effective balance between performance, interpretability, and scalability.

#### 2.2.1.3 AutoEncoders

Autoencoders, nestled within the landscape of unsupervised learning, emerge as a powerful tool for learning compact and meaningful representations of data [44]. Positioned alongside other traditional machine learning methods, autoencoders stand out for their ability to capture latent features and intrinsic structures within the data without the need for explicit labels [44]. This is achieved through a process of encoding the input data into a lower-dimensional representation, followed by decoding to reconstruct the original input. By minimizing the reconstruction error between the input and output, autoencoders learn to extract essential features and patterns from the data, facilitating tasks such as data denoising [45], dimensionality reduction [46], and anomaly detection [47]. Their flexibility and generative capabilities make autoencoders invaluable across a spectrum of applications, including image and text data processing, recommendation systems, and anomaly detection in cybersecurity. Moreover, the emergence of deep learning architectures has further propelled the advancement of autoencoders, enabling the exploration of more complex and expressive representations. In summary, autoencoders play a vital role within the broader landscape of machine learning, offering a powerful framework for unsupervised learning and feature learning tasks, driving innovations across various domains [48].

#### 2.2.1.4 Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNNs) stand as a cornerstone within the landscape of machine learning, particularly in the domain of deep learning. Positioned alongside other neural network architectures, CNNs are specifically tailored for processing grid-like data, such as images and audio spectrograms. CNNs leverage convolutional layers to automatically learn hierarchical representations of features within the input data, capturing spatial patterns and local dependencies through shared weight filters [49]. This unique architecture allows CNNs to excel in tasks such as image classification [50], object detection [51], and semantic segmentation [51], where understanding spatial relationships and extracting intricate features are crucial [48]. Their ability to learn directly

from raw data without the need for explicit feature engineering has revolutionized fields such as computer vision and image processing. Furthermore, CNNs exhibit scalability and generalization capabilities, enabling effective transfer learning across different domains and applications. In essence, Convolutional Neural Networks represent a fundamental component of the deep learning landscape, offering unparalleled capabilities in processing and understanding complex structured data, and driving innovations across a myriad of domains.

#### 2.2.1.5 Long-Short Term Networks (LSTM)

Long Short-Term Memory (LSTM) networks represent a crucial advancement within the landscape of recurrent neural networks (RNNs), particularly in the domain of sequential data processing. Positioned alongside other deep learning architectures, LSTM networks are designed to address the vanishing gradient problem encountered in traditional RNNs [52], enabling them to effectively capture long-range dependencies and temporal patterns within sequential data. This unique architecture allows LSTM networks to retain and selectively update information over extended time periods, making them well-suited for tasks involving time-series data, natural language processing, and speech recognition [53]. By incorporating memory cells with gated units that regulate the flow of information, LSTM networks can learn and retain relevant information over multiple time steps, facilitating more accurate predictions and capturing complex temporal dynamics [54]. As a result, LSTM networks have become indispensable tools in various applications, including language translation [55], sentiment analysis [56], and stock market prediction [57]. Their ability to model sequential data with long-term dependencies positions LSTM networks as essential components within the broader landscape of machine learning, driving advancements in understanding and processing sequential information.

#### 2.2.1.6 Transformers

The Transformer model represents a groundbreaking advancement within the landscape of machine learning, particularly in the realm of natural language processing (NLP) and sequence modeling. Positioned alongside other deep learning architectures, the Transformer architecture has revolutionized the field by introducing a novel mechanism for processing sequential data [58]. Unlike traditional recurrent neural networks (RNNs) and convolutional neural networks (CNNs), Transformers rely on self-attention mechanisms to capture long-range dependencies and contextual relationships within the input sequence [59]. This unique architecture enables Transformers to process entire sequences in parallel, allowing for efficient computation and scalability [60]. With its ability to capture contextual information effectively, Transformers have achieved remarkable success in various NLP tasks, including language translation, text summarization, and question answering [61]. Moreover, Transformers exhibit versatility and applicability beyond NLP, with extensions and adaptations for tasks such as image generation, graph processing, and recommendation systems. In summary, the Transformer model represents a transformative milestone within the landscape of machine learning, offering unparalleled capabilities in processing sequential data and driving innovations across diverse domains.

### 2.2.2 Dynamic Programming

Dynamic Programming is a powerful optimization technique, which will be leveraged later on in the paper, and is used to solve complex problems by breaking them down into smaller subproblems [62]. Unlike traditional divide-and-conquer approaches, dynamic programming systematically stores solutions to subproblems in a table or array, allowing for efficient reuse of previously computed results. This method is particularly useful for problems with overlapping subproblems, where the same subproblems need to be solved multiple times. By solving each subproblem only once and storing the results, dynamic programming significantly reduces computational complexity and improves overall efficiency [62]. Common applications of dynamic programming include

solving optimization problems, such as finding the shortest path in a graph (e.g., Dijkstra's algorithm) [63] or maximizing profit in a resource allocation problem (e.g., the knapsack problem). Dynamic programming algorithms offer a systematic and efficient approach to solving complex problems, making them indispensable tools in algorithm design and optimization.

### 2.2.3 Automatic Deviation Detection - State of the Art

Several machine learning techniques, including those based on artificial neural networks (ANN) [64] and AutoEncoders [65], have been deployed in the aviation industry to analyze flight data and predict anomalies. These techniques are capable of providing binary responses, indicating whether an anomaly is detected or not. However, a notable limitation of these approaches is their inability to pinpoint the specific issue causing the anomaly, especially if it is process-specific. While they effectively flag deviations from normal behavior, they fall short in offering detailed insights into the underlying root causes of anomalies. This lack of granularity poses challenges for aviation maintenance and operations teams, as it hampers their ability to perform targeted troubleshooting and implement corrective measures. As a result, there is a growing need for machine learning models that not only detect anomalies but also provide actionable insights into the specific processes or subsystems affected, enabling more effective maintenance and operational decision-making in the aviation sector.

Automatic process deviation has been in other domains as well. Using a model derived from actual behavior traces, Lu et al. performed deviation detection and identified common and uncommon behaviors in a business process [66]. Yang et al. approached a problem in deviation detection in complex medical processes, by combining both data-driven and knowledge-driven techniques [67]. Their approach was split up into different steps, dealing with hierarchical model structures. These model structures would have parent tasks containing a series of subtasks that would need to be performed. By manually recording the actions performed by medical staff through video footage, they were able to create action sequences. The first step in their process would be to annotate each action to its corresponding parent task. Once these new annotated sequences have been

made, they would be aligned with a reference sequence in order to find the best possible split. From this point, they would process using a Conformance Checking algorithm readily available in the ProM software suite. This algorithm is capable of returning errors of commission, and omission, though this computation is very computationally demanding and would frequently cause computers to run out of memory (Dell, Windows 10 OS, Intel Xeon 3.7 GHz CPU, 48GB RAM), while also requiring all sequences to be in the form of Petri Nets (containing an activity trace of the same workflow). Further processing enabled Yang et al. to determine scheduling errors using a secondary algorithm, as the ProM conformance algorithm only dealt with commission and omission errors.

Christov et al. performed deviation detections in processes for chemotherapy and blood transfusion through the creation of knowledge-driven workflows [68]. They simulated the error detections within the process by using synthetic activity traces (achieved by inserting artificial process errors).

**Our approach, akin to Yang et al., seeks to bridge the gaps between data-driven and knowledge-driven models by integrating them**. Data-driven checking may better interpret observed data, at the cost of having sufficient samples. On the other hand, knowledge-driven methods have no dependence on data; though can become exponentially difficult to make as the complexity of the system increases. We can use machine learning aspects to understand nuanced relations between activities, while leveraging the workflow model to identify these problematic tasks.

## 2.3 Error-State Prediction

Error-State Prediction is an emerging field within aviation safety that focuses on forecasting conditions in which pilots are more prone to committing errors [69, 70]. While traditional approaches primarily concentrate on detecting real-time mistakes, this forward-looking approach aims to anticipate potential error-prone states before they occur. By leveraging predictive analytics and machine learning techniques, Error-State Prediction seeks to identify subtle cues and patterns in pilot behavior, environmental factors, and physiological indicators that may indicate an increased likelihood of errors [71, 72].

This proactive strategy holds significant promise for enhancing aviation safety by allowing preemptive interventions or adjustments to mitigate risks before they escalate. Moreover, the integration of Brain-Computer Interfaces (BCIs) offers a novel avenue for gathering real-time cognitive and physiological data from pilots, enabling more accurate and timely error-state predictions [73–75]. As technology continues to advance, Error-State Prediction represents a promising frontier in aviation safety research [69, 70], with potential applications extending to other high-stakes domains beyond aviation, such as healthcare [71], and industrial maintenance [76, 77].

### 2.3.1 Brain-Computer Interfaces (BCI)

BCIs have been growing in popularity with regards to scientific research in neurotechnology [78]. These BCI devices can come in multiple forms, from non-invasive, and partially invasive to invasive, based on how close electrodes get to brain tissue [79]. Electroencephalograms (EEG), a form of non-invasive BCI devices, can read the user's brain electrical activity through a mesh of electrodes. This electrical activity can then be interpreted to understand subject brain states along with their brain connectivity [80]. One such study mapped connectivity of cognitive monitoring and executive control of strategy adjustment to the anterior cingulate cortex (ACC) through neuroimaging and brain potential studies [81]. These monitoring processes are heavily intertwined with cognitive task performance and can be indicative of performance degradation. It is believed that the ACC can detect potential errors or conflicting responses prior to being consciously aware of them [82] and then begins to signal the need to regulate future uncertainty and risk minimization. An example of this could be a student answering a question on an exam for which he is uncertain of the response. The ACC will detect potential errors or conflicting responses, such as misinterpretations of questions and then activate adaptive control processes, prompting the student to adjust their approach, clarify uncertainties, and optimize their performance to minimize the risk of errors to enhance overall exam success. Ridderinkhof was able to identify these markers using the Flanker test and an EEG device. They concluded that errors were observed to be presaged by a distinct pattern of electrophysiological brain activity on the trial preced-

ing the error and that the ACC serves to indicate the need to engage control processes (including response inhibition) to minimize the risk of errors [83].

## 2.4 Pre-Error State Detection - State of the Art

Predicting perceptual errors in BCIs is still in a novel state with Batmanova et al claiming to have taken the first steps towards predicting these error states. They setup a perceptual decision-making task to collect behavioral data and brain activity signals [84]. This experiment required participants to perceive 400 stimuli in the form of neckar cubes [1] continuously with a brief pause interval, attempting to replicate real-life situations where decision making would be done in a stressful environment. To increase the likelihood of errors they used ambiguous stimuli while also decreasing the stimulus exhibition time, resulting in a subject error rate close to 13%. Their machine learning stack involved using two one-dimensional convolutional neural networks (CNN) in series. The first CNN performed an EEG channel wise convolution, while the second performed a time-point-wise convolution, with the size output being a hyperparameter, K, that was selected based on model performance. Their EEG data was taken two seconds before the stimulus and two seconds after the stimulus, filtered by performing z-score normalization and thresholding at a value of 1. Despite receiving exceptional results, an F1-score of 88% and accuracy of 92.6%, there is no mention of transferability of the model beyond the dataset. They also mention a limitation of not addressing participant leakage, where a single participant's data may be found in both training and validation set. The error and success trials of a participant should be locked in either the test or validation set, if not, results can be skewed by providing exceptional F1 and accuracy scores, as the paper demonstrates.

A secondary study explores the use of EEGs to measure human decision confidence levels [85]. A visual perceptual decision confidence experiment involves 14 participants

1. A Necker cube is an optical illusion consisting of a simple wireframe drawing of a cube. It is a classic example of an ambiguous figure, where the 3D structure of the cube can be perceived in two different ways. The viewer can see the cube oriented with either one of two faces as the front face, and the perception can spontaneously switch between these two interpretations.

performing a task while EEG data is recorded. The task involves showing blurred images of animals to the participants, while they must decide for the animal group, then report their confidence levels about the selection from one through five. The study treats measuring decision confidence as a pattern classification task, using two classifiers (Support vector machine with RBF kernel and a deep neural network ANN with shortcut connections to retain original information) trained with EEG features. Their results show EEG signals can assess decision confidence, achieving peak accuracy of 49.14% and F1-score of 45.07% when validating all five confidence levels, and 91.28% accuracy with an average F1-score of 88.92% for extreme confidence levels (confidence of 1 and 5, while ignoring confidence levels in between). Instead of using raw EEG, they passed power spectral density (PSD), differential entropy (DE), differential asymmetry (DASM), rational asymmetry (RASM) and asymmetry (ASM) features into their machine learning models, with DE performing the best. Though model generality is important for real time applications such as flight where cases of confidence between 2 and 4 are quite common in real world applications, which may be problematic for model performance.

Another study claiming to be the first online study in real car decoding driver's error-related brain activity introduces an EEG-based BCI designed to decode error-related brain activity for use in driving assistance systems [86]. Conducted in both a car simulator (N = 22) and a real car (N = 8), participants received directional cues before approaching intersections. The study classified error-related potentials from EEG using Linear Discriminant Analysis as a supervised learning algorithm to predict whether the cued direction aligned with the driver's intention. Offline experiments achieved an average classification accuracy of $0.698 \pm 0.065$ in the simulator and $0.682 \pm 0.059$ in the real car, both significantly above chance level. Online tests showed equivalent performance in simulated and real car driving, supporting the BCI's feasibility for decoding signals and estimating driver intention in real-world driving scenarios.

# CHAPTER 3

# METHODOLOGY

In this chapter we will discuss the various methodology of the deviation detection component and the pre-error state detection component, beginning with the experimentation for dataset acquisition.

## 3.1 Data Acquisition

Experiments were conducted following the ethical certificate F20-CERSES-2925 Pilot AI to assess real-time cognitive activity in pilots during takeoff in an Airbus A320 simulator. The participants, consisting of seven pilots, including five A320 pilots, and six engineers with expertise in aircraft maneuvers, were divided into two groups experiencing different takeoff scenarios to eliminate learning bias. The participants had to release the parking brake, do a takeoff procedure, and climb until 3000 ft without using autopilot. Cognitive workload (CW), heart rate (HR), and pupil diameter (PD) were measured using an EEG headset, Polar H10 heart rate monitor strap, and Gazepoint GP3 eye tracker. A total of 136 takeoffs were performed by the 13 participants, who had an average age of 36 years, 604 flight hours, and 8.5 years of piloting experience [87].

### 3.1.1 Setup

The experiment adhered to a strict procedure approved by both partners and the ethics committee. Participants were provided with a detailed guide on the A320 takeoff procedure prior to the experiment, so that they may familiarize themselves with the simulator's handling. On the day of the experiment, participants underwent a 30-minute familiarization flight to practice taking off with the simulator. They were also briefed on the various failure procedures they might encounter during this familiarization. An experimenter guided participants through the Airbus A320 takeoff process, ensuring conformity. After this, measurement tools were set up, and the actual experiment commenced.

Participants were not informed of the scenarios beforehand, intentionally increasing cognitive workload. The experiment consisted of two 20-minute sessions: one with failures and one without. In the session with failures, participants were aware that failures could occur during takeoff but were unaware of the type or timing of the failures. Additionally, they were not informed of the weather conditions before each scenario. The 6 Scenarios can be seen below in table 3.I.

| Scenario | Time | Weather | Failure |
|---|---|---|---|
| 1 | 1:45 pm | No wind, no clouds | No |
| 2 | 6:00 am | Clouds at 2700 ft, rain | No |
| 3 | 9:00 pm | No wind, no clouds | No |
| 4 | 5:30 am | No wind, no clouds | Yes, EF at 80 knots |
| 5 | 6:00 am | 15 knots crosswind | Yes, EF at 140 knots |
| 6 | 6:00 am | Low visibility, rain | Yes, EF at 80 knots |

Table 3.I – An overview of the different XPlane A320 scenarios and their corresponding Engine Failures (EF).

### 3.1.2 EEG Logging

Central to EEG is the precise placement of electrodes on the scalp, which allows researchers and clinicians to record and analyze neural signals with high temporal resolution. EEG electrode placement follows standardized protocols to ensure consistency across studies and facilitate comparisons between different subjects and research findings.

The placement of EEG electrodes is based on the international 10-20 system, which provides a standardized framework for positioning electrodes relative to anatomical landmarks on the scalp. The "10-20" refers to the distances between electrode placements, which are either 10% or 20% of the total front-to-back and right-to-left distances of the skull. This system divides the scalp into regions denoted by letter-number combinations, such as Fp (frontopolar), F (frontal), C (central), P (parietal), and O (occipital), with odd-numbered electrodes located along the midline and even-numbered electrodes positioned laterally [88].

The specific electrode placements used in EEG recording depend on the research objectives, experimental design, and regions of interest within the brain. For example, frontal electrodes (F) are commonly used to monitor cognitive processes and emotional responses, while central electrodes (C) are positioned over sensorimotor areas to capture motor activity and somatosensory processing. Parietal (P) and occipital (O) electrodes are often employed to study visual processing and attentional mechanisms [89].

In addition to the standard 10-20 system, alternative electrode placements and configurations may be used for specialized applications or to target specific brain regions. For example, high-density EEG arrays with a larger number of electrodes can provide finer spatial resolution and more detailed brain mapping. These arrays may include additional electrodes to cover specific cortical areas or to capture signals from deep brain structures.

In the conducted Pilot experiment, the EEG headset chosen for data acquisition was the 16 Channel OpenBCI model. This specific EEG headset is equipped with 16 electrodes strategically positioned across the scalp to capture electrical signals generated by neuronal activity, as seen in figure 3.1. The OpenBCI headset is known for its versatility, user-friendly design, and compatibility with various research and application scenarios.

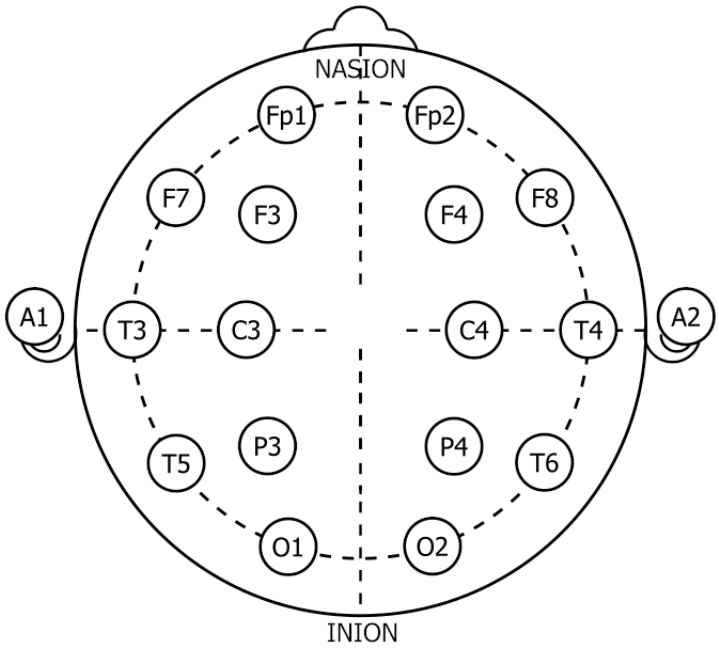


Figure 3.1 – 16-Channel EEG channel placements, with additional A1 and A2 ear lobe clips from OpenBCI.

## 3.2 Ontological Reference Model for Piloting Procedures

When performing a deviation analysis, an action sequence needs to be compared to a baseline sequence for validation. The Ontological Reference Model for Piloting Procedures, which we will reference as the Aircraft Ontology, contains pilot tasks and actions for procedures in various phases of flight [90]. This Aircraft Ontology focuses primarily on takeoff for the time being, and has been created by referencing standardized procedures from Airbus flight crew operating manual in circumstances including standard takeoff, loss of an engine, rejected takeoff, and other forms of takeoff. Current implementation of this ontology contains takeoff procedures as well as procedures for specific takeoff events. This ontology is provided as owl files, which can be loaded in python by using the owlready2 python package. Owlready2 is a library for working with ontology-oriented programming, which allows you to create, manipulate, and query ontologies in various formats such as OWL, RDF/XML, and RDFS. It provides a high-level API for ontology management and integrates with various reasoning engines for advanced

querying and inference [91]. We can then use these ontologies to search for all Task Objects and Constraint objects. Once a Task object has been identified, we can query the hasNbConstraint and hasConstraint parameter to identify a task's execution conditions. A task's corresponding actions can also be queried by using the hasNbAction and hasAction parameter. These actions and constraints are utilized recursively and are compared with the simulation environment to validate whether a task can be executed. The ontology hierarchically organizes its objects into phases (e.g., takeoff), then tasks, and finally actions, which is a structure that can be effectively utilized in the deviation analysis.

## 3.3 Simulator API

The deviation analysis needs to be able to gather information from the simulated environment in order to perform its assessment. The simulator selected for this project is named XPlane, running on Windows, using the A320 Ultimate aircraft package. However, it is important to have the option of extending the usability of this deviation assessment beyond one single type of simulator. This design choice led us to creating an Application Programming Interface (API) that has the ability to accommodate different types of simulators. The API is developed in C++, to leverage its computational efficiency, and communicates with XPlane using a User Datagram Protocol (UDP) connection. Access to the API is possible through any python application by simply using pip to install the XPlaneApi package. This architecture can then be used to retrieve data from XPlane, or the simulator of choice. The high-level architecture of the simulator API is illustrated in figure 3.2

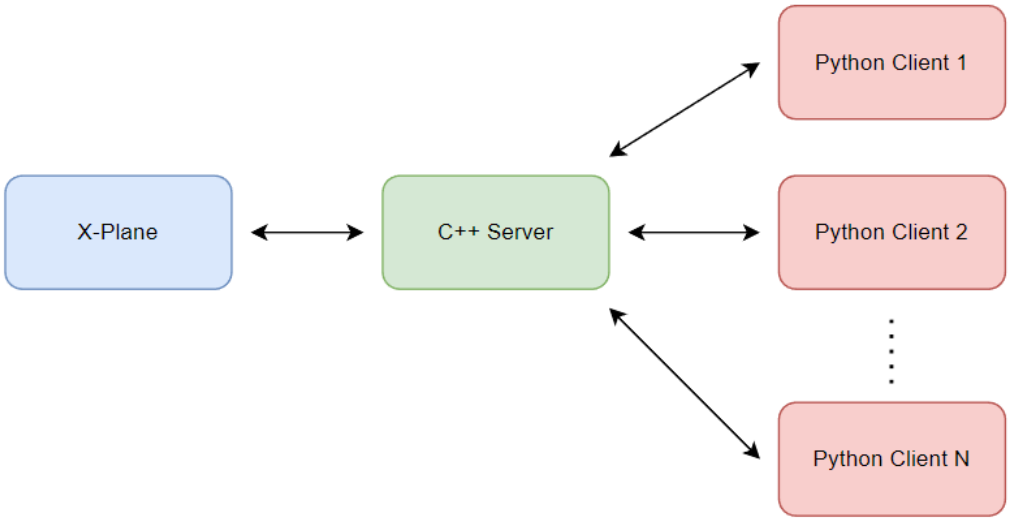


Figure 3.2 – X-Plane simulator API Architecture

With the goal of having a multi-simulator capable system, the API is loaded with a Subscriptions.yaml file, containing labels with their associated tags. Each tag contains the simulator specific data reference string, which represents a specific aircraft system within the simulator. A snippet of the Subscriptions.yaml file can be seen in figure 3.3, where the LeftThrustLever label is the variable name that API clients will use to identify the system. Furthermore, the tag *sim/cockpit2/engine/actuators/throttle_ration[0]* is the tag that XPlane uses to identify the thrust lever for engine 1. This system enables applications that communicate with the API to continuously reference the thrust lever for engine 1 as the label name, while the simulator specifications can be swapped out as needed in yaml files. The frequency in Hz at which a label can be polled from the simulation can also be selected. Increasing the frequency for highly used systems such as pedals, or sidesticks, while decreasing the frequency for infrequently used knobs and buttons will reduce the load on the system (if a large number of labels are being observed).


Figure 3.3 – Labels, tags, and frequency in the Subscriptions.yaml for XPlane-API

## 3.4 API - C++ Server

In this section we will discuss the different aspects and design structures of the C++ API Server.

### 3.4.1 LibXPlane-Udp-Client Cross-Platform Compatibility

The C++ server was built around a public library named libXplane-udp-client, who's public github can be found at https://github.com/dotsha747/libXPlane-UDP-Client. The only issue was that this library was not written with cross-platform compatibility, meaning that it was only supported on linux. This was an issue as the project was scoped with a single windows computer node, in which X-Plane would be running. As a result, this repository was forked and reworked with the aforementioned cross-platform compatibility. Libraries such as winsock2 were used for windows socket manipulation.

### 3.4.2 IPC (Inter-Process Communication) Protocol ZeroMQ

Once the libXplane-udp-client Windows compatible version was ready for deployment, the server needed to be programmed to accept incoming communication from python clients. There are several ways that this task can be accomplished, but the importance here is that whichever protocol is selected is also freely available on Python.

ZeroMQ is an open source universal messaging library (also known as ØMQ, 0MQ, or zmq) used by several large tech firms such as Microsoft, Facebook, and Samsung [92]. It can accomplish various types of messaging protocols through sockets such as N-to-N with patterns like fan-out, pub-sub, task distribution, and request-reply. It is also multi-platform if ever the API needs to be ported to another platform.
ZeroMQ was selected as the main source of communication between the C++ server and the python clients. The pub-sub model was selected, which enables each client to have a publishing and subscribing socket between the server for back-and-forth communication.

### 3.4.3 Architecture of C++ Server

The C++ Server begins by launching the X-Plane Beacon object. This object is used to detect the presence of X-Plane. Once X-Plane has booted up and the aircraft has spawned (users are able to interact with the cockpit), the beacon listener will indicate to the server that X-Plane is alive and ready to receive subscriptions and publications.
The server will then proceed to initialize the X-Plane UDP client, which was imported previously, to interact with X-plane Datarefs [1] and Commands. Subscriptions to X-Plane can be placed in the *Subscriptions.yaml* file. These subscriptions are placed side-by-side with an update frequency in Hz. This update frequency is used by X-Plane to check a Dataref for changes. This C++ Server architecture can be seen below in figure 3.4.

1. "Dataref" (short for data reference) is a variable that allows plugins, scripts, and other components to read or write data within the simulator. Datarefs are essential for interacting with X-Plane's internal state, controlling aircraft systems, or displaying information on custom instruments.

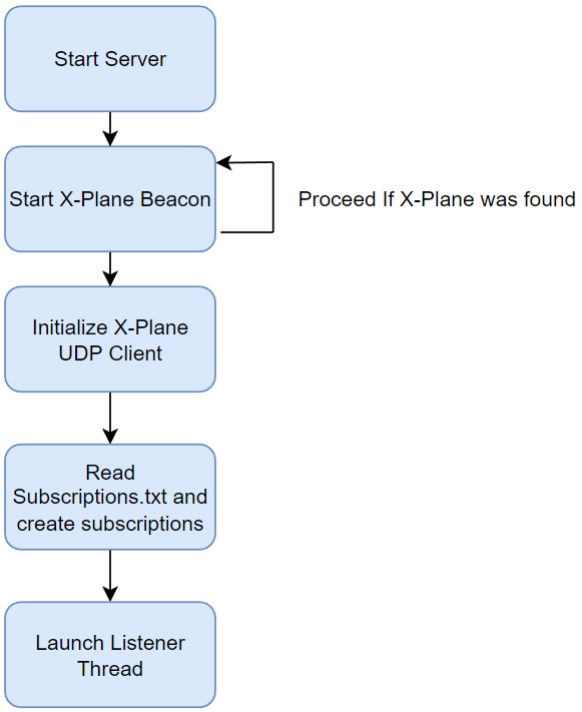


Figure 3.4 – C++ API Server Core Architecture

### 3.4.4 Subscription Storage

Each time the C++ Server receives an updated dataref from the specified subscriptions list, it is stored into a map value to be referenced at a later time. This map can be read at any time by a python client who wishes to retrieve information. The update frequency of each map entry is dictated by the frequency specified in the subscriptions.yaml.

### 3.4.5 Listener Thread

As illustrated in figure 3.5, once the listener thread is launched, the C++ Server binds itself onto local ports 5555 (as a subscriber) and port 5556 (as a publisher). These ports are important for Python clients who wish to make themselves known to the Server. Once the server receives a message from the Python client, the port manager within the Server locates available ports and opens a dedicated communication thread with the

connected client. This thread can be used to communicate with the Python client, and send information that is requested by the client.

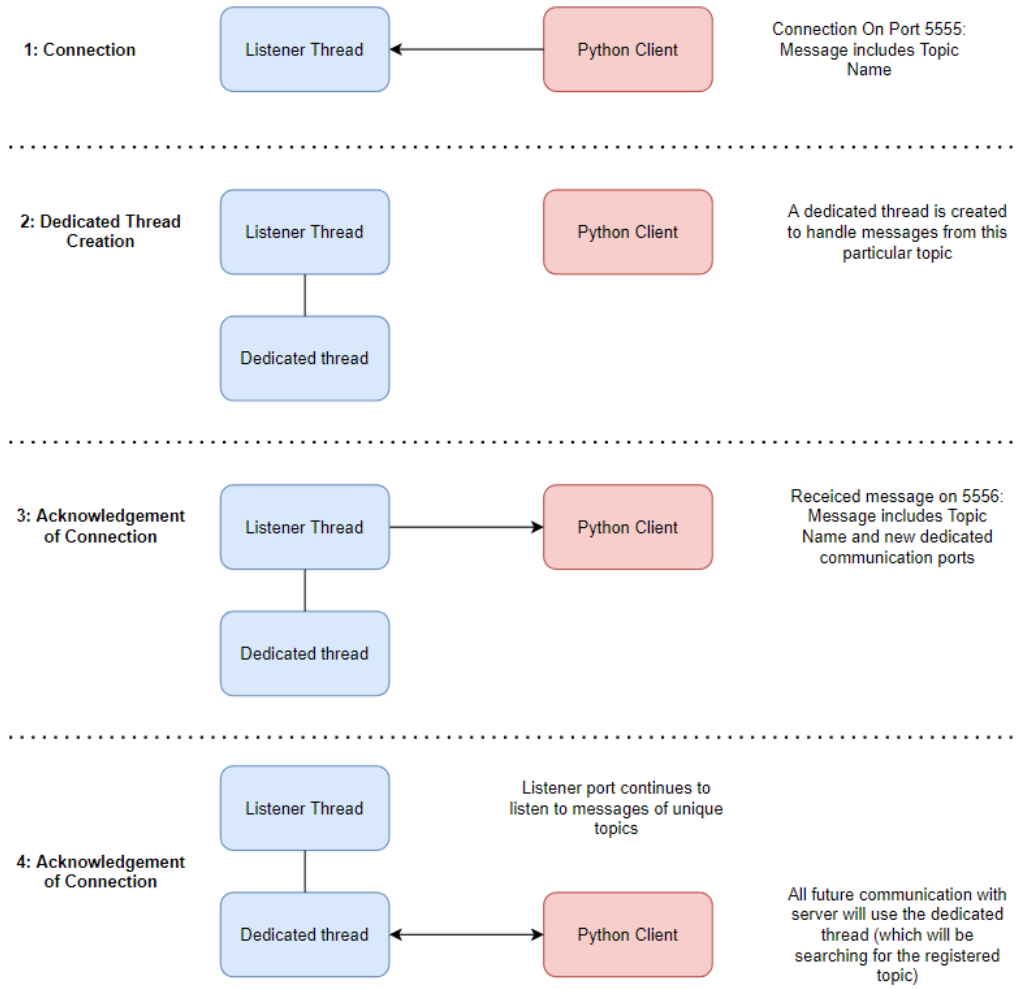


Figure 3.5 – Listener Thread Connection Sequence

### 3.4.6 Dedicated Thread

Once the dedicated thread has established a connection with the Python Client, a listening loop is setup to wait for messages received by a particular topic. Each UDP message contains 4 parts – [Topic, Requested Command, Dataref, Value]. In table 3.II we can see the description of each message parameter.

| **Message Part** | **Description** |
|---|---|
| Topic | The topic of registration for the Python Client |
| Requested Command | The Requested command tells the server what the client wishes to do with the Dataref of interest. At the time of writing there are 5 possible commands that could be interpreted by the Server. Connection, Disconnection, Read, Set, and Command. It should be noted that for all commands except for Set, the value parameter of the message part should be 0. |
| Dataref | The Dataref of interest. |
| Value | The value that the Client wishes to set the dataref. |

Table 3.II – Message Part descriptions.

## 3.5 API - Python Client

In this section we will discuss the different aspects and design structures of the API Python Client.

### 3.5.1 Python Client Connect and Disconnect

The connection function creates a publisher and subscriber on the C++ Server Listener ports, and sends an initial four part multipart UDP packet containing the Client's Topic, the "Connection" command, and 0 values for the dataref and it's value. The subscriber then waits for a response back, which will contain the new port for the dedicated thread. Publisher and Subscriber threads are then re-binded to the new ports and the connection function exits successfully.

Upon Disconnection, the Client sends a four part multipart UDP packet containing the Client's Topic, the "Disconnection" command, and 0 values for the dataref and it's value. This enables the Server to close the thread and liberate the dedicated sockets so that another Client may connect.

### 3.5.2 Python Client Functions

In table 3.III a list and description of all available functions in the API can be found. These functions can be utilized to communicate with XPlane.

| Functions | Description |
|---|---|
| getDataRef | Retrieves a value for a given dataref. The multipart message contains the topic of the client, the "read" command, the specified dataref, and a value of 0 for the value (as it will not be read). The response received is in the form of a utf-8 string, meaning numerical values need to be casted. |
| setDataRef | Set the value of a particular dataref. All values must be string values, meaning that numerical values need to be initially casted to strings before being passed into this function. The function returns a boolean to indicate whether the Server has successfully received the request. |
| sendCommand | Use X-Plane commands with the cockpit. Once again, the function returns a Boolean value to indicate whether the Server has successfully received the request. |

Table 3.III – Python Client API functions.

## 3.6 API Installation

In this section we will discuss procedures on how to install the API C++ server and the python client.

### 3.6.1 C++ Server Installation

The latest and greatest releases of the C++ server can be found in the github repository of the project. From here the latest revision of the binaries can be downloaded, unpacked, and executed from inside its containing folder. It should be self contained and requires no additional steps.

### 3.6.2 Python Client Installation

The Python Client was designed with ease in mind. For this reason, it was setup with a PyPi repository, which enables users to import this API into their python code with relative ease. All that is required is to perform a pip install of the package: pip install XPlaneApi. Then the API can be imported with the following import: from XPlaneApi import XPlaneClient. For more installation instructions, the PyPi can be accessed at the following link, with additional installation instructions found on the front page.

## 3.7 Runner

The Runner provides the main running utilities that the Deviation Model requires to gather information and remain in sync with the simulator. As illustrated in figure 3, upon initialization, the Runner will load the Aircraft Ontology dictionary (using the owlready2 ontology package) into a data object, that unpacks all required data, while also creating additional structures to facilitate data access during simulation time (such as querying for next tasks and previous tasks in sequences). Once all libraries are loaded, the Runner initializes the simulator API with its corresponding subscriptions file, and connects to it as a client. The successful connection to the API allows the Runner to commence the simulation loop, which runs at a frequency of 10Hz.

Once the simulation loop begins, the Runner can run in Execution Mode or in Observation Mode. As the names suggest, execution mode will follow the sequence of takeoff tasks present in the Aircraft Ontology and execute them in the XPlane cock-pit. Observation mode, on the other hand, will observe the cockpit and record all per-formed actions. Labels are assigned to the action sequences according to their predicted task, encoding in the form taskNumber+actionNumber. A completion flag is also appended onto these actions to ensure that they have been completed according to the task specification. For example, if the expected next task in the sequence re-quires both throttles to be engaged to 50% and the cockpit polls a 40% placement, then this task will be recorded as started – but not completed. These actions will then be passed onto the Deviation Model for sequence assessment, which can be seen as the yellow block in figure 3.6.

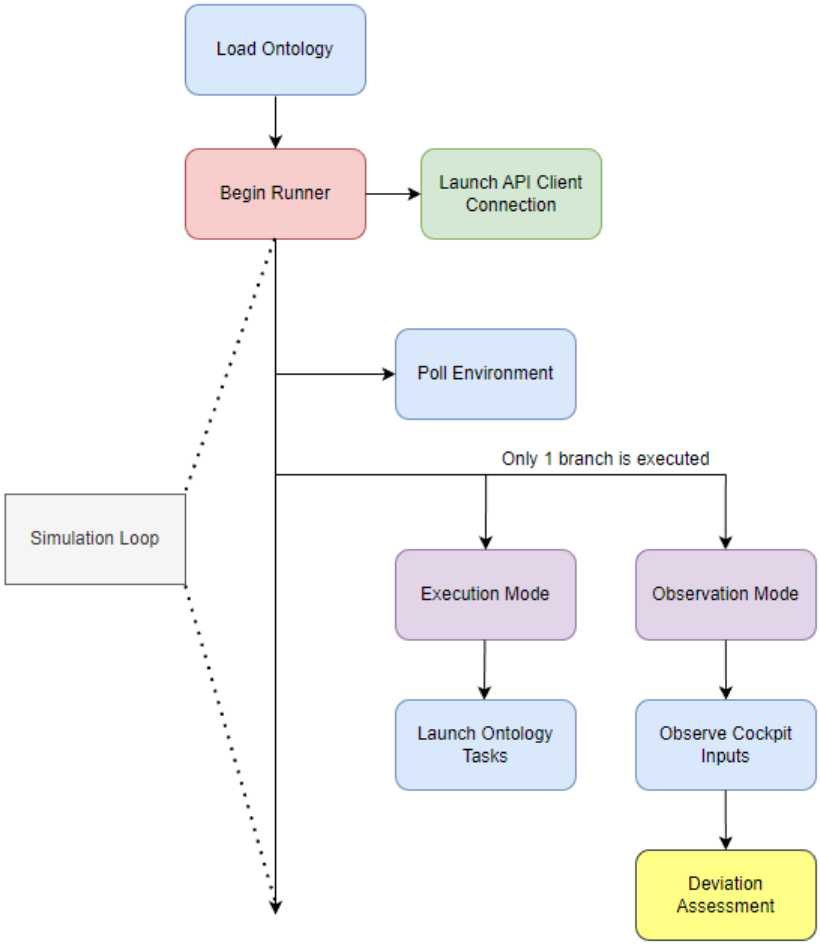


Figure 3.6 – XPlane Runner execution diagram.

## 3.8 Deviation Model Architecture

The Deviation Model receives the action sequences from the cockpit, which we label as our Pilot Actions, and those from the Aircraft Ontology, referred to as our Reference Actions. We have divided the deviation assessment into four main steps as follows:

### 3.8.1 Deviation Step 1

As seen in figure 3.7, the tasks are hierarchically organized into phases, then tasks, and finally actions. Phases, such as takeoff, may contain several tasks, each with their associated action.

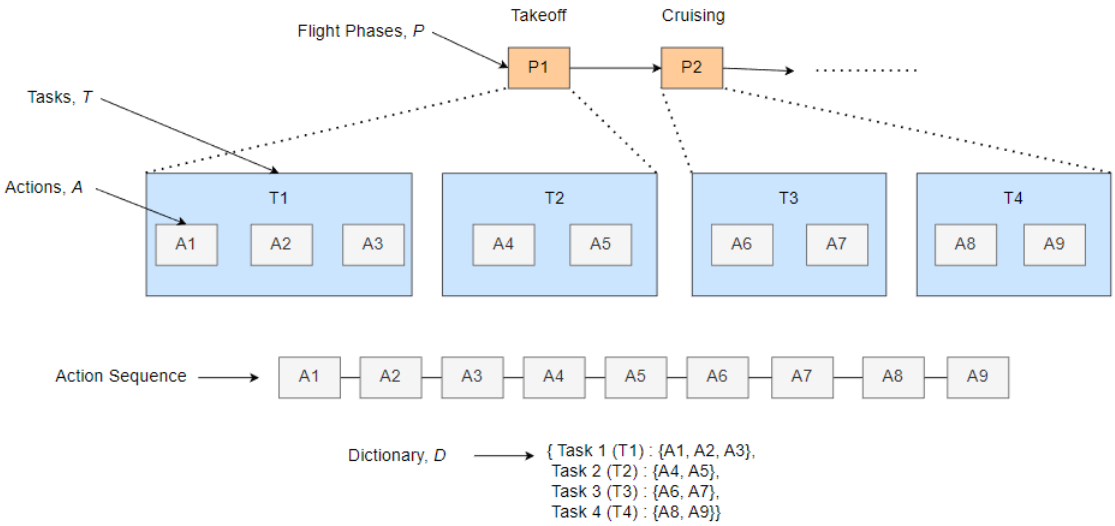


Figure 3.7 – Hierarchical organization of phases, tasks, and actions.

An example of a task includes setting the throttles, which contains two separate actions of setting the throttle of engine 1 and engine 2. The action sequence that is received by the Deviation Model contains actions that have been polled from the cockpit by the Runner. In this first step, actions are annotated using the Task Dictionary, which contains a dictionary of tasks with their associated actions. This process can be seen in figure 3.8 which contains the overview of the deviation process, as actions are annotated to their corresponding tasks (Pilot actions 1, 2 and 4 belonging to task 1, and actions 3, 5, and 6 belonging to task 2). Task actions should always be adjacent to each other, as they should be done in sequence with each other. After annotating the Reference and Pilot actions in figure 4 step 1, we can already start seeing where sequences are not fully lined up, which leads us to identifying problematic tasks.


Figure 3.8 – Deviation Model Overview

### 3.8.2 Deviation Step 2

These newly annotated sequences now need to be aligned together before being able to validate which tasks are out of place. The Needleman-Wunsch global alignment algorithm is a commonly used global alignment algorithm, which has been adopted in many sequence alignment software, and will be the choice of algorithm for this deviation assessment [93]. This algorithm is frequently employed in computational biology

and bioinformatics for DNA or protein sequencing [94]. It utilizes dynamic programming to determine the ideal global alignment between two sequences. The algorithm for the Needleman-Wunsch is outlined in figure 3.9.

**Algorithm 7:** Needleman-Wunsch Global Alignment Algorithm

**Input:** Sequences $seq1$ and $seq2$, match score $matchScore$, mismatch penalty $mismatchPenalty$, gap penalty $gapPenalty$
**Output:** Aligned sequences $alignedSeq1$ and $alignedSeq2$, alignment score

```
n ← length(seq1)
m ← length(seq2)
# Initialize the score matrix and traceback matrix
scoreMatrix ← array of size (n + 1, m + 1)
tracebackMatrix ← array of size (n + 1, m + 1)
# Initialize the first row and column of the score matrix
for i ← 0 to n do
    scoreMatrix[i][0] ← i × gapPenalty
    tracebackMatrix[i][0] ← "up"
for j ← 0 to m do
    scoreMatrix[0][j] ← j × gapPenalty
    tracebackMatrix[0][j] ← "left"
# Fill in the score matrix and traceback matrix
for i ← 1 to n do
    for j ← 1 to m do
        match ← scoreMatrix[i − 1][j − 1] + (matchScore if seq1[i − 1] ==
          seq2[j − 1] else mismatchPenalty)
        delete ← scoreMatrix[i − 1][j] + gapPenalty
        insert ← scoreMatrix[i][j − 1] + gapPenalty
        scoreMatrix[i][j] ← max(match, delete, insert)
        if scoreMatrix[i][j] == match then
            tracebackMatrix[i][j] ← "diag"
        else if scoreMatrix[i][j] == delete then
            tracebackMatrix[i][j] ← "up"
        else
            tracebackMatrix[i][j] ← "left"
# Traceback to get the aligned sequences
alignedSeq1 ← ""
alignedSeq2 ← ""
i ← n
j ← m
while i > 0 or j > 0 do
    if tracebackMatrix[i][j] == "diag" then
        alignedSeq1 ← seq1[i − 1] + alignedSeq1
        alignedSeq2 ← seq2[j − 1] + alignedSeq2
        i ← i − 1
        j ← j − 1
    else if tracebackMatrix[i][j] == "up" then
        alignedSeq1 ← seq1[i − 1] + alignedSeq1
        alignedSeq2 ← " − " + alignedSeq2
        i ← i − 1
    else
        alignedSeq1 ← " − " + alignedSeq1
        alignedSeq2 ← seq2[j − 1] + alignedSeq2
        j ← j − 1
return alignedSeq1, alignedSeq2, scoreMatrix[n][m]
```

Figure 3.9 – Algorithmic description of Needleman-Wunsch global alignment algorithm

### 3.8.3 Deviation Step 3

Once the alignments have been received from step 2, they are superimposed, and processed so we can extract problematic tasks along with their actions. Once sequence gaps are identified, they can easily be classified into added, omitted, and out of sequence. The algorithm for this process is described below in figure 3.10.

**Algorithm 1:** Deterministic Deviation Assessment

**Input:** Reference Sequence $S_R$, Pilot Sequence $S_P$, Action-Task dictionary $D$
**Output:** Labeled Problematic Actions $A_{Labeled}$
1. Annotate actions with Task labels: $S_R^P = Annotate(S_R, D)$, $S_P^P = Annotate(S_P, D)$
2. Insert anchor action at start of both sequences $S_R^{P\prime}, S_P^{P\prime} = Insert(A_{Anchor}, index = 0)$
3. Perform Needleman-Wunsch alignment to receive 2 alignment sub-traces $t_1, t_2$
4. Generate Gapped Sequence $S_G$ (containing gaps wherever problems found) **for** $i = 1$ **to** $len(t_1)$ **do**
    **if** $t_1[i] == t_2[i]$ **then**
        store action into $S_G$
    **else**
        store a gap ' ' into $S_G$
5. Extract indices $I_G$ of gapped entries in $S_G$
6. Gather added actions $A_{Added}$ from subtraction of unique entries from $S_P$ and $S_R$
7. Gather omitted actions $A_{Omitted}$ from subtraction of unique entries from $S_R$ and $S_P$
8. Loop through each index in $I_G$ to find out of place actions $A_{OutOfPlace}$
**for** $i \in I_G$ **do**
    **if** $S_R[i + shift]$ *in* $A_{Omitted}$ *and* $S_P[i + shift] == S_R[i + shift]$ **then**
        shift $-= 1$
        continue
    **else**
        **if** $S_P[i + shift]$ *in* $S_R$ **then**
            append action into $A_{OutOfPlace}$
        **else**
            continue
9. Label and combine $A_{Added}$, $A_{Omitted}$, $A_{OutOfPlace}$ to receive $A_{Labeled}$
10. return $A_{Labeled}$

Figure 3.10 – Algorithmic description of deterministic deviation assessment

### 3.8.4 Deviation Step 4

According to the Aircraft Ontology, certain tasks may be executed simultaneously, such as tasks which describe checking the takeoff N1, gradually releasing the sidestick, and monitoring the primary flight display for the Takeoff phase. These tasks can be

recorded in different orders continuously by the Runner due to their tandem nature. Siamese Long Short-Term Memory (LSTM) networks are a type of recurrent neural network architecture that is often used for tasks that involve measuring similarity or distance between two input sequences. It uses identical LSTM networks with shared weights to process each sequence, and then combines the resulting hidden states to make a prediction. Siamese LSTM networks have been applied to a wide range of tasks, and had a lot of success in identifying these hidden relations between sequences in several domains including text similarity, image matching, and speaker verification [95–97]. As illustrated in figure 7, the pilot and reference sequence are passed through a Siamese LSTM that we have trained on 6607 generated piloting sequences (correct and incorrect pairings). If the Siamese LSTM indicates that the sequence is the same as the reference sequence, then the flagged tasks will be recorded as acceptable deviations and will be placed in an ignored table for future iterations. The Siamese LSTM, as seen in figure 3.11, was trained using a learning rate of 0.001, LSTM input size of 300, hidden size of 100, and 2 layers.

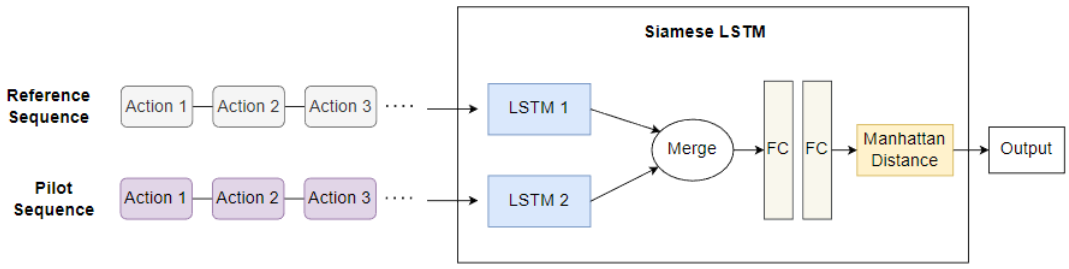


Figure 3.11 – Siamese LSTM structural overview

## 3.9 Pre-Error-State Detection

In this section we will discuss the methodology of the pre-error signal detection system. We will examine the dataset creation and the different models that were experimented in the creation of the system.

### 3.9.1 Flanker Dataset

The cognitive brain controller interface dataset, aptly named COG-BCI, was used to train the machine learning models, as it is a standardized and highly regulated dataset containing flanker EEG and behavioral data [98]. The Flanker task is a choice reaction task designed to induce errors and conflict during binary decisions. In its arrowhead version, participants are presented with stimuli consisting of 5 horizontal arrows, where they must respond to the middle arrow while disregarding flanking arrows. As seen in figure 3.12, flanker stimuli can be congruent (flanking arrows point in the same direction) or incongruent (flanking arrows point in the opposite direction to the central arrow). Each trial involves a 2000ms inter-stimulus interval (ISI) followed by a 16ms display of the stimulus. Stimuli are presented equally frequently in a pseudorandom order, and participants respond by indicating the target direction with keyboard keys. Feedback about trial outcomes is provided, and the task involves 120 trials (30 for each stimulus type), lasting about 10 minutes. The structure of the test can be seen in figure 1. Participant responses, error rates, and reaction times are recorded throughout the task. Instructions are given before the run begins. The dataset was compiled over three sessions, each separated by one week.

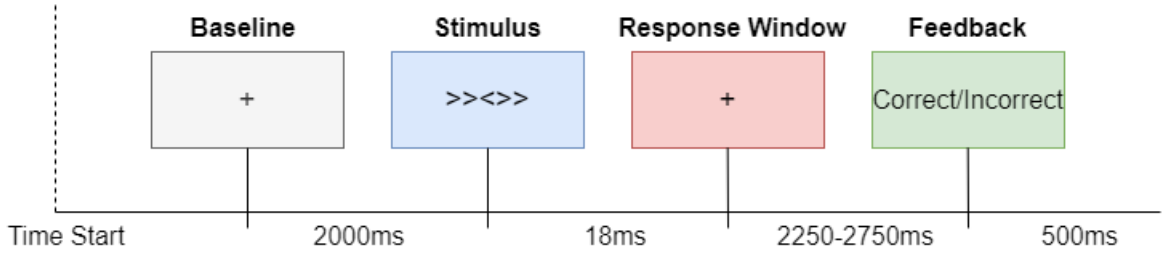


Figure 3.12 – Flanker test procedure and timing windows.

### 3.9.2 EEG Preprocessing and Dataset Creation

The EEG signals underwent a comprehensive preprocessing pipeline to enhance their quality and prepare them for subsequent analysis. Initially, we applied a high-pass finite

impulse response (FIR) filter with an automatic filter length, set to a cutoff frequency of 1 Hz, using the Python MNE toolbox, to eliminate the average component of the EEG signal and isolate the desired frequency bands. Following this, a zero-phase notch filter (or band reject filter) was used to reject signals at the 50Hz frequency, using the FIR filter design 'window' method. Subsequently, we down sampled the frequency from the original 500Hz to 250Hz to optimize computational efficiency. To address potential artifacts such as heartbeat and eye blinks, we performed Independent Component Analysis (ICA). An epoch, representing a time sequence of 1 second before and 0.5 seconds after the stimulus signal, was extracted for each trial. Every individual epoch underwent visual inspection, and any instances with lingering large amplitude artifacts were removed. Choosing a z-score threshold of 5 during the rejection procedure was intended to balance the retention of a sufficient number of samples while identifying and discarding EEG spikes that exhibited extreme deviations from the expected distribution, ensuring effective artifact removal without overly stringent criteria. The dataset creation overview can be seen below in figure 2. After completing the rejection procedure, our dataset comprised of 1153 erroneous samples and 8521 non-erroneous samples, ensuring a robust and refined dataset for further analysis and model development. It should be noted that training and validation sets were created with no participant leakage (Single participant data would not be found in both training and validation set). This overall dataset creation process can be visualized in figure 3.13.

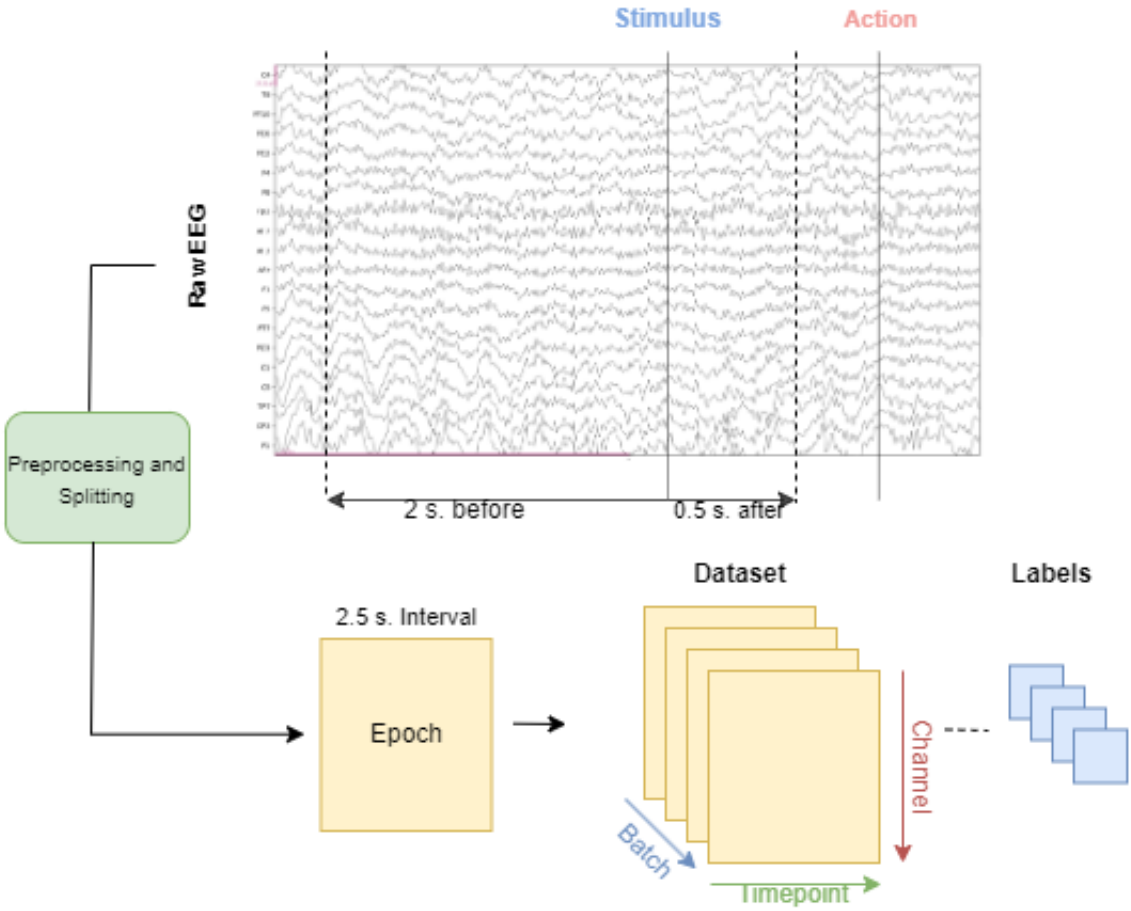


Figure 3.13 – EEG processing pipeline to create labeled data from 64-channel Flanker data (not all channels can be seen in the graph illustrated).

### 3.9.3 Adjusting Imbalanced Datasets

Imbalanced datasets refer to situations where the distribution of classes is uneven, with one or more classes having significantly fewer instances than others. In our COG-BCI dataset, the occurrence of erroneous data points is significantly lower compared to the correct ones. Addressing imbalanced datasets is crucial as it can lead to undersampling is not a feasible choice. As a result, we explored two oversampling techniques: Synthetic Minority Over-sampling Technique (SMOTE) [99] and Random Oversampler [100], with the latter demonstrating superior performance. SMOTE is a widely used

method for generating synthetic samples to balance class distribution by interpolating between existing minority class samples. On the other hand, Random Oversampler randomly replicates minority class samples to achieve a balanced dataset.

### 3.9.4 Models

Several machine learning models were used on the dataset to compare performance. The following will describe these models and the hyper parameters chosen for optimal performance in the Flanker task. It should be noted that hyperparameter tuning techniques such as Cross Validation and Grid Search from SKLearn and open-source library Optuna were used in order to find optimal configurations for the models.

#### 3.9.4.1 SVM and Random Forest models.

Support Vector Machines (SVM) and Random Forests are both types of supervised machine learning algorithms. They fall under the category of classification algorithms, as they are commonly used for tasks where the goal is to predict the class or category of a given input based on labeled training data. SVM is particularly known for its effectiveness in binary and multi-class classification, while the Random Forests model is an ensemble learning method that can be used for both classification and regression tasks. They are both relatively easy to implement and have been proven to provide great results on EEG data [101, 102]. These models serve as a great starting point in terms of identifying trends in our EEG dataset.

Support Vector Machines (SVM) work by finding the optimal hyperplane that maximally separates data points of different classes in a high-dimensional space. The algorithm aims to maximize the margin, which is the distance between the hyperplane and the nearest data point of either class, leading to better generalization. SVMs can handle both linear and non-linear depending on the kernel function selected. We decided to use an RBF kernel function, which is commonly used with EEG data. The parameters chosen for the SVM model can be found in table 3.IV.

| Parameter | Value |
|---|---|
| C | 1.0 |
| Kernel | RBF |
| Gamma | Scale |
| Probability | True |
| Shrinking | True |
| Tol | 0.001 |

Table 3.IV – SVM Parameters

Random Forest models operate by constructing multiple decision trees during training and outputting the mode of the classes (classification) or the mean prediction (regression) of the individual trees. Each tree is built using a random subset of the training data and a random subset of features, reducing overfitting and improving generalization. The combination of diverse trees results in a robust and accurate model. The parameters chosen for the Random Forests can be found in table 3.V:

| Parameter | Value |
|---|---|
| Bootstrap | True |
| Ccp_alpha | 0.0 |
| Criterion | Gini |
| Max_depth | 50 |
| Max_features | 0.7 |
| Min_samples_leaf | 1 |
| Min_samples_split | 10 |
| N_estimators | 100 |

Table 3.V – Random Forests parameters

#### 3.9.4.2 Double Convolution

Convolutional Neural Networks (CNNs) are a type of deep learning architecture, specifically designed for tasks involving structured grid-like data, such as images. CNNs are particularly powerful for image classification, object detection, and image segmentation tasks. While SVM and Random Forests are traditional machine learning algorithms that often require handcrafted feature engineering, CNNs automatically learn hierarchical features directly from the raw input data through the application of convolutional lay-

ers. To process the dataset through CNNs we follow similar architecture to Batmanova et al. with a few modifications by performing the following steps (also illustrated in figure 3.14):

1. We first take our Channel by Time-point matrix from our batch and perform a 1-dimensional convolution across the channel axis. This convolution slides along the dimension of the EEG channels, while keeping the time-point dimension fixed, performing an averaging of EEG amplitude at every moment.
2. A second 1-dimensional convolution is performed across the remaining 1-dimensional time dimension (where each point is an average across all channels) resulting in a 1-dimensional vector with k features. This value k is then treated as a hyperparameter that can be tuned to improve model performance. Naturally, with too few k values, the model will not efficiently learn, while with too many k values, the model will have redundant features.
3. We then perform a 1-dimensional batch normalization to improve training stability and convergence speed. The batch normalization is also useful to mitigate the vanishing and exploding gradient problems during backpropagation.
4. A dropout layer is added with a dropout rate of 0.3. This regularization technique helps to break tight coupling between neurons, reducing the risk of overfitting and improving the model's ability to generalize to unseen data.
5. Finally, two fully connected layers with hidden dimensions of size 1024 are used to perform the classification. A leaky-Relu activation is performed in between these two layers with a sigmoid activation function for the binary classification output.

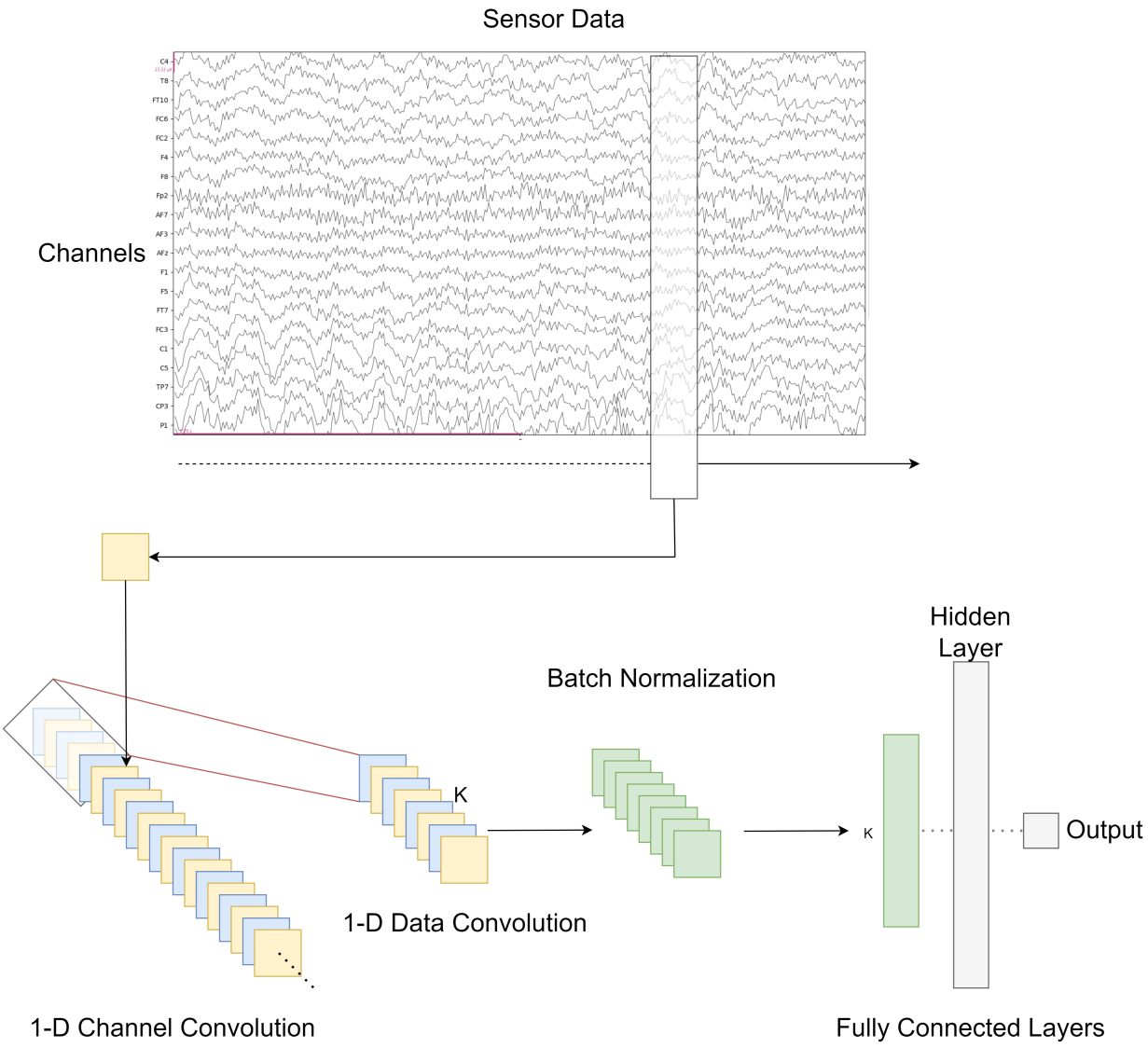


Figure 3.14 – Illustration of the double convolution neural network structure from raw Flanker EEG data (64 Channel - not all channels displayed in image). It should be noted that 'K' is equivalent to the variable 'k' mentioned in the text.

**Double Convolution Hyperparameters**

The following are hyper parameters that were modified in search of the best performant CNN model:

- *Initializers* are methods used to set the initial values of the model's weights and biases. We experimented with the following two initialization techniques: Xavier Uniform, and Random Uniform, with the former providing the best results.
- *Learning Rate* determines the size of the steps taken during the optimization process, influencing the convergence and stability of a model. The learning rate range searched was between 0.00001 and 0.01. A final learning rate of 0.00671 was used for optimal model performance.
- *Epochs* refer to the number of times a model processes the entire training dataset during training. Each epoch consists of forward and backward passes, allowing the model to update its weights and learn from the data iteratively. The epoch range searched was between 10 and 100, with 27 epochs providing the best results.
- *Batch size* represents the number of training examples utilized in a single iteration during model training. It influences the efficiency and memory requirements of the training process, with larger batch sizes often providing computational speed-ups but potentially leading to reduced generalization. The batch size range that was searched was between 32 and 700, with 314 providing the best results.
- *Activation functions* introduce non-linearities to the model's output, allowing it to learn complex relationships and patterns in the data. The following activation functions were used: Relu, Elu, Leaky Relu, Selu, with Leaky Relu providing the best results.
- *Optimizers* are algorithms used to minimize the error or loss function during the training of a neural network by adjusting the model parameters. They play a crucial role in updating the weights of the network in a way that facilitates convergence, ensuring the model learns effectively from the training data and generalizes well to new, unseen data. The following optimizers were used in the

training of this model: Adam, and Adaptive Gradients (Adagrad), with the latter providing the best results.

- $k$ value represents the number of features that will be passed to the fully connected layers after the two convolutional layers. The range that was searched for this value was between 50 and 100, with 79 providing the best results.

**Impact of Double Convolution on EEG Frequency Bands**

The preprocessing steps involving double convolution significantly influence the frequency content of the EEG signal.

- **Channel-wise Convolution:** This step involves averaging the EEG signals across the channels, resulting in a single averaged EEG signal.
- **Time Series-wise Convolution:** The second step involves averaging over the time dimension within temporal sub-windows of width $N/K$.

Both steps effectively act as low-pass filters, attenuating higher frequency components of the EEG signal. Specifically, with a sampling frequency of 250 Hz and a signal duration of 2.5 seconds (625 samples), averaging over windows of 8 samples significantly reduces the effective sampling frequency, thereby attenuating frequencies above approximately 30 Hz.

Given the preprocessing steps, higher frequency bands such as alpha (7-13 Hz), beta (15-25 Hz), and gamma (35-45 Hz) waves are significantly attenuated. This results in the retention of lower frequency components, primarily in the delta (1-3 Hz) and theta (4-6 Hz) bands. Empirical experimentation with various values of $k$ demonstrated that extreme values (either approaching 1 or the maximum possible value) resulted in significant performance losses. This suggests that the optimal performance of the network is achieved when the preprocessing steps retain key frequency components in the delta and theta bands. This validates prior work demonstrating that error-related brain activity appears in the theta band [103, 104].

#### 3.9.4.3 Transformer

Transformers are a type of deep learning architecture that has gained significant popularity, especially in natural language processing tasks. Unlike traditional sequence-based models, transformers are designed to process sequences of data in parallel, making them highly efficient for tasks involving sequential data, such as language understanding, translation, and text generation. Transformers are known for their self-attention mechanism, allowing them to capture relationships between different elements in a sequence, making them particularly powerful for handling long-range dependencies. The transformer architecture was initially proposed for natural language processing but has been successfully applied to various other domains, including computer vision, speech processing, and EEG [18]. The architecture that will be used for this task was inspired by the Conformer architecture conceived by Song et al. The architecture, as seen in figure 3.15, can be split up into three main parts as follows:

- **Part 1** - Convolution Module: The convolution module is designed by decomposing the two-dimensional convolution operator into two one-dimensional layers, separating temporal and spatial convolutions, much like the double convolution used previously. The first layer employs k kernels performing convolutions along the time dimension. The second layer consists of k kernels acting as a spatial filter to capture interactions between electrode channels. Batch normalization is applied for enhanced training and reduced overfitting, while exponential linear units (ELUs) serve as the activation function for nonlinearity. The third layer performs average pooling along the time dimension, smoothing temporal features to prevent overfitting and decrease computational complexity. The resulting feature maps are rearranged, with the electrode channel dimension squeezed and the convolution channel dimension transposed, enabling the feeding of all feature channels of each temporal point as tokens into the next module.
- **Part 2** - Self-Attention Module: The Self-Attention Module is introduced to capture global temporal dependencies in EEG features, utilizing self-attention to enhance the decoding of context-dependent representations within low-level

temporal-spatial features. Tokens from the previous module are linearly transformed into query (Q), key (K), and value (V), and their correlations are evaluated using dot product, with a scaling factor to prevent vanishing gradients. The resulting attention score is obtained through a Softmax function and applied to V with a dot product. This process is repeated N times in the self-attention module, incorporating a multi-head strategy to enhance representation diversity. The multi-head attention results are concatenated, and the entire attention computation is performed N times.

- **Part 3** - Classifier Module: Finally, the Classifier Module employs two fully connected layers to output an M-dimensional vector after a Softmax function, using binary cross-entropy as the loss function for the entire framework.

As a result, we have a model that processes EEG data through temporal and spatial convolution layers, arranges them into tokens with a pooling layer, applies self- attention layers, and uses fully connected layers for classification results.

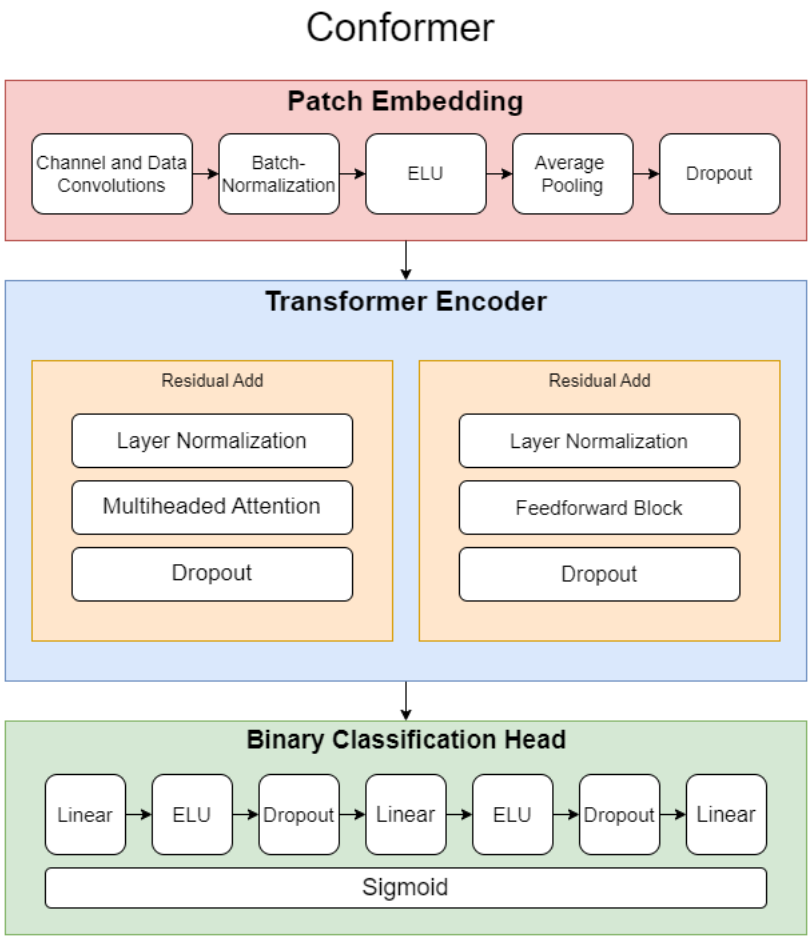


Figure 3.15 – Illustration of the three components contained within the conformer structure.

### Transformer Hyperparameters

Similarly, to the CNN model, the following are hyper parameters that were modified in search of the best performant transformer model:

- *Initializers*Xavier Uniform, and Random Uniform, with Xavier Uniform providing the best results.
- *Learning Rate* Between 0.00001 and 0.01. A final learning rate of 0.0072 was used for optimal model performance.
- *Epochs* Between 10 and 150, with 30 epochs providing the best results.
- *Batch size* Between 32 and 100 due to hardware limitations, with 75 providing the best results.
- *Optimizer Scheduler* is a mechanism that dynamically adjusts the learning rate during training. It helps optimize the convergence of the model by controlling how much the model's parameters are updated in each iteration, allowing for better performance and faster convergence. The following schedulers were used: ReduceLROnPlateau, StepLR, CosineAnnealingLR, CosineAnnealingWarmRestarts, CyclicLR, with CyclicLR providing the best results cycling between 0.001 and 0.01.
- *Optimizers* Adam, Stochastic Gradient Descent (SGD), and Adaptive Gradients (Adagrad), with a tuned Adam optimizer providing the best results using beta-1 value of 0.285 and beta-2 value of 0.927.
- *k* Between 10 and 100, with 20 providing the best results.

# CHAPTER 4

# RESULTS

## 4.1 Deviation Detection Results

In order to assess a preliminary evaluation of the proposed deviation detection method, this model was applied to the takeoff procedure defined in the Aircraft Ontology. Sequences of steps were generated based on the Ontology's specifications and were mutated to represent problematic process executions. This process is referred to as Mutation Testing, and is a software testing technique used to evaluate its effectiveness by measuring its ability to detect changes or mutations. The types of mutations that have been introduced to the system are deletions (removal of an action), insertions (adding an action), substitution (replacing an action with another), and swap (switching the places of two actions).

We first performed a random walk through the Aircraft Ontology and retrieve a take-off sequence. Each individual action is then encoded with a unique identification number. The deviation model then consumes this sequence and outputs the problematic tasks with a label that indicates if the action is out of place (1), has been add-ed (2), or omitted (3). An example of each type of mutation is performed on the action sequence that is input into the model and the model's respective outputs can be seen in figure 4.1. The outputs for these cases demonstrate that the deterministic assessment of the deviation model performs as expected.

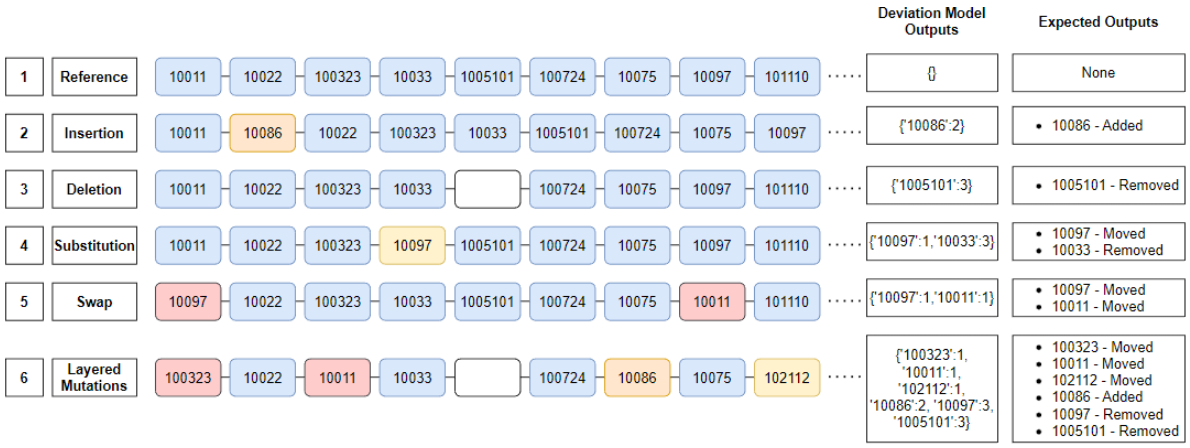


Figure 4.1 – Mutation testing on synthetic action sequences with fixed order.

The actions 102112, 102011, and 102314 in figure 4.2 represent respectively the actions of checking the takeoff N1, gradually releasing the sidestick, and monitoring the primary flight display. These are tasks that are meant to be completed at the same time, and should not be returned as problematic if they would be completed in an order that is not represented in the reference sequence. In order to replicate a scenario where these three actions are executed out of order, we perform a swap mutation on these actions alone. This swap mutation would trigger the deterministic deviation assessment and would return these tasks as problematic due to the fact that they no longer match the reference sequence. The Siamese LSTM, however, successfully captures the relationship between these tasks through trained examples and overrides the problematic labels by indicating that there are no deviations – because these tasks can be performed in any order with respect to each other. We can therefore validate our first hypothesis by confirming that the deviation assessment is capable of detecting several types of problematic actions, while also understanding the relation between certain tasks and ignoring them if necessary [105].

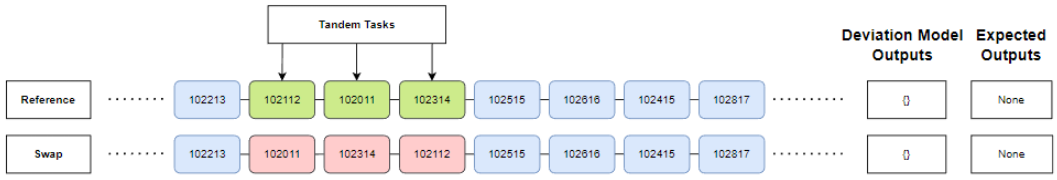


Figure 4.2 – Mutation testing on synthetic action sequences that can be executed in any order.

Furthermore, deviation assessments using this method are extremely fast at 0.0179 seconds for an analysis sequence with length of 23 actions (for one takeoff sequence walk). When increasing the analysis length to 70 actions, the model increased in time to 0.0499 seconds (an average increase of 0.001 seconds per additional action), which is still well within the bound of a 10Hz (0.1 seconds) simulation time requirement. Times were taken on Windows 10 computer using i7-5930K 3.50GHz processor. If we compare these values to the 70.5 average sequence length at 5.98 seconds by Christov et al. [68] for their real time medical deviation, we have achieved a substantial increase in assessment speed. However, computer specifications were not defined in their work and could affect cross comparisons. We can then validate our real-time requirements by confirming that this assessment can be completed at frequency of 10Hz for a maximum estimated 120 actions per phase.

### 4.1.1 Simulation Testing

Using XPlane as the simulation platform and involving 5 different non-pilot users performing 6 takeoffs from the scenarios outlined in the experimentation description, a total of 22 perceived errors were recorded. These perceived errors encompassed a variety of deviations from standard procedures and task executions within the simulated flight environment. The deviation detection system successfully identified and flagged all 22 perceived errors, demonstrating its efficacy in real-time error detection.

Upon closer examination, it was observed that out of the 22 flagged errors, 16 instances were determined to be deviations that were correctly suppressed by the LSTM-

based deviation detection system. These deviations, although sequentially diverging from standard procedures, were identified as nuanced variations within acceptable bounds, highlighting the system's capability to distinguish between critical deviations requiring intervention and minor discrepancies that could be safely ignored.

In response to the identified errors and deviations, the users were provided with real-time corrective feedback and guidance. This immediate feedback mechanism allowed for timely correction of errors during the simulation sessions, enabling users to adjust their actions and adhere to prescribed procedures more effectively. By incorporating real-time corrective interventions, the deviation detection system not only identified deviations but also facilitated continuous learning and improvement among users, enhancing their proficiency in executing takeoff procedures within the simulated flight environment.

### 4.1.2 Limitations

Despite promising results, there are, however, still some limitations to this technique when it comes to training the Siamese LSTM. A sufficient amount of distributed data sequences is required to train this model, along with clever data engineering for known relational tasks. Even though the sequence chains are taken directly from the Aircraft Ontology, results might change if pilot cockpit behavior varies drastically, which may add complexity. These behavior changes could also introduce mutation types other than the ones that have been synthetically tested (deletions, insertions, substitutions, and swaps).

## 4.2 Pre-Error State Detection Results

### 4.2.1 Flanker Data Pre-Error State Detection

The results from the analysis of the Flanker dataset using the models are presented in table 4.I. It should be noted that accuracy values are misleading to the dataset imbalance. With an error rate close to the 13% mark, a model that has learned to simply pick the most abundant class without actually classifying between the two will achieve a high accuracy. For this reason, we have accumulated Macro-Averaged metrics for Pre-

cision, Recall, and F1 scores. Macro-averaged metrics calculate Precision, Recall, and F1 score for each class independently and then average them. This ensures that each class is given equal importance in the evaluation, addressing the imbalance and providing a more nuanced understanding of performance [106]. A Macro-Averaged F1 score above 0.6 would indicate a classifier with moderate to good performance, while above 0.8 indicating excellent performance. Additionally, we have included the area under the receiver operating characteristic (AUC ROC), which serves as a metric that quantifies the overall performance of a binary classification model by measuring the area under the curve formed by plotting the true positive rate against the false positive rate across various classification thresholds. A value of 0.5 for the AUC ROC is indicative of a random classification with no discriminative power, with values closer to 1 presenting a strong discriminative power.

We can see that the simpler machine learning algorithms, namely SVM and Random Forests, receive high accuracy values but underperform in all other metrics. Their classification often seems random or biased towards one class, indicating that the complexities in the EEG data might not be suited for these particular classifiers. The double convolution model begins to discern between the two classes; however, its performance still remains quite low and contains a lot of false negatives. The transformer model manages to outperform the other models by a significant margin. We see a reduction in the number of false negatives and a final F1 of 0.610, which serves as promising signs of its classification power for erroneous states. It is interesting to note that a channel reduction from 62 to 16 increased the performance of the classifier. This performance increase could be due to a decrease in attenuation of higher EEG frequencies, allowing the model to capture more significant information. The 16 chosen electrodes followed the standard 10-20 system distribution and not simply a random subset of the original 62 channels.

We can see the generated validation graphs below in figure 4.3 from the training process of the Conformer architecture. There is a graph for each Test Accuracy, Test F1, Test Kappa score, Test Precision, Test Recall, and the validation loss.

| Dataset | Model | Acc. (%) | AUC ROC | *Precision | *Recall | *F1 |
|---|---|---|---|---|---|---|
| **Flanker** | SVM (Linear) | 70.86 | 0.559 | 0.519 | 0.533 | 0.512 |
| | SVM (RBF) | 80.48 | 0.523 | 0.493 | 0.493 | 0.491 |
| | Random Forests | **86.30** | 0.484 | 0.439 | 0.500 | 0.468 |
| | Double Convolution | 83.88 | 0.591 | 0.583 | 0.555 | 0.563 |
| | Conformer (62 Channel) | 84.08 | 0.633 | 0.599 | 0.600 | 0.600 |
| | Conformer (16 Channel) | 82.40 | **0.652** | **0.611** | **0.610** | **0.610** |

Table 4.I – Model results on Flanker Dataset. Asterix (*) indicates a macro averaging for the given value.

Below in figure 4.4 we can see the ROC curve, and also the associated AUC value. An ROC AUC of 0.65 suggests moderate discriminative ability of our binary classification model. This value indicates that the model performs better than random guessing but may not be highly accurate in distinguishing between the positive and negative classes.

Taking a look at the confusion matrix in figure 4.5 we can get a better idea of the inference capabilities of our model.

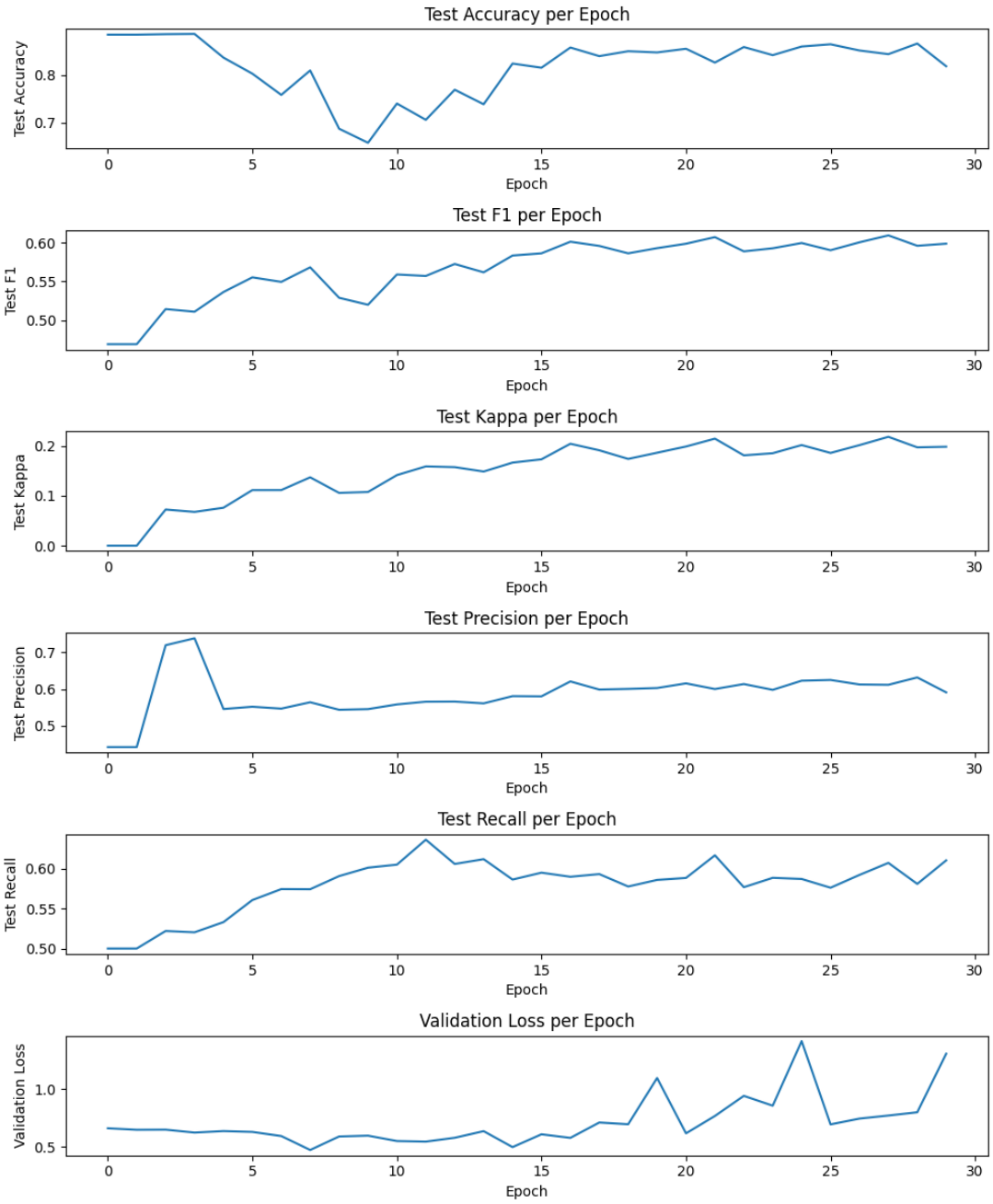


Figure 4.3 – Test Accuracy, Test F1, Test Kappa score, Test Precision, Test Recall, and the validation loss for Conformer architecture with 16 channel EEG dataset.

**It should be noted that a label of 0 indicates a pre-error state, while a label of 1 indicates a non-pre-error state.**

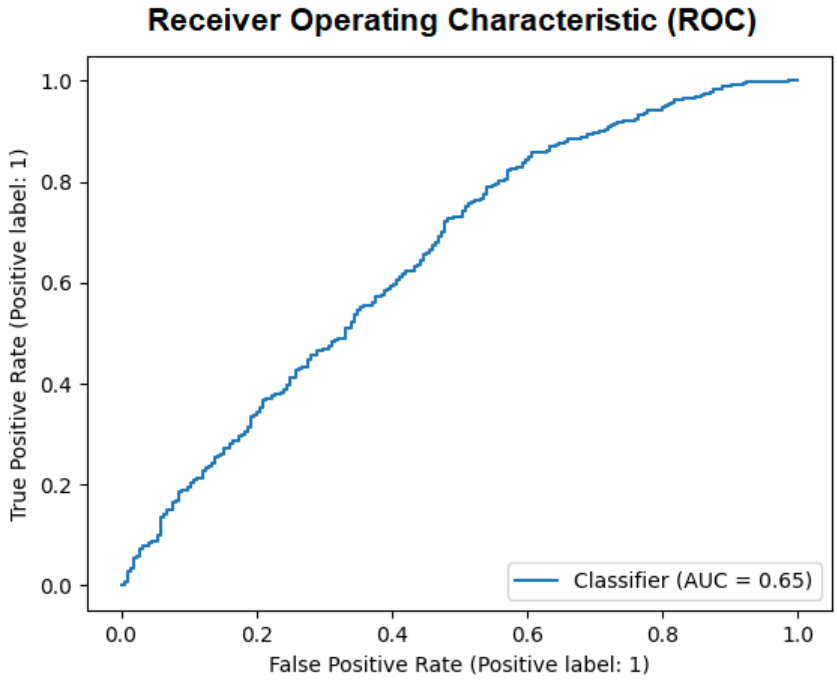


Figure 4.4 – ROC AUC of the Conformer architecture with 16 channel EEG dataset.

The true positive count of the confusion matrix (value of 80) indicates that the model correctly identifies some positive instances of pre-error states, albeit at a lower rate compared to false positives. While the true positive rate is an essential measure of a model's performance, it's crucial to consider how it balances with false positives in the context of the application.

The model has a relatively high number of false positive predictions (144), indicating that it frequently misclassifies negative instances as positive. This suggests that the model may be overly sensitive or prone to making type I errors, which could lead to false alarms or incorrect identifications of positive cases.

With 216 false negative predictions, the model misses a significant number of positive instances. This high false negative rate suggests that the model may be conservative or overly cautious in predicting positive cases, potentially leading to missed opportunities for correctly identifying relevant instances. In our case, this value is of less importance as a pilot who is not incapacitated being flagged will not necessarily be prone to

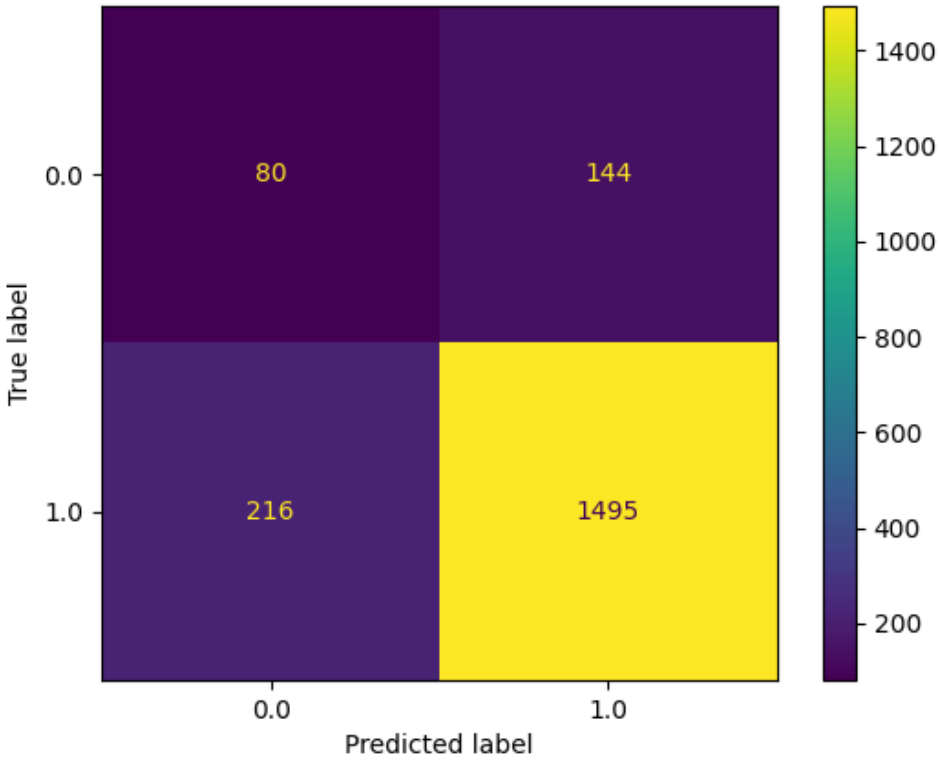


Figure 4.5 – Confusion Matrix of the Conformer architecture with 16 channel EEG dataset.

making more mistakes.

Finally, the true negative count (1495) indicates that the model correctly identifies a large proportion of negative instances. High true negative rate is desirable, especially in scenarios where correctly identifying negative cases is crucial.

This discrepancy can be a testament to the human guessing of a binary choice test. Participants that were uncertain of the result could have correctly guessed the answer, resulting in a false positive entry. This type of result would require additional test parameters put in place, such as confidence level measure after an input, in order to accurately gauge.

### 4.2.2 Pilot Data Pre-Error State Detection

While many studies conclude their experiments at this juncture, our objective extends beyond, aiming to assess the model's applicability in a secondary domain - substantiating the transfer-ability of the classifiers. To achieve this, we have collected EEG data from aviation pilots and intend to evaluate the classifiers' discriminative power specifically during takeoff procedures [107]. We can examine the performance of these models in table 4.II, where we can see that the performance reduces slightly, but performs relatively as expected. The transformer model still outperforms the other models in terms of classification with a macro-averaged F1 above random classification, which is a promising indicator.

| **Dataset** | **Model** | **Acc. (%)** | **AUC ROC** | ***Precision** | ***Recall** | ***F1** |
|---|---|---|---|---|---|---|
| **Pilot** | SVM (Linear) | 78.81 | 0.499 | 0.498 | 0.499 | 0.499 |
| | SVM (RBF) | 82.68 | 0.486 | 0.423 | 0.486 | 0.452 |
| | Random Forests | **88.42** | 0.498 | 0.443 | 0.498 | 0.469 |
| | Double Convolution | 77.38 | 0.532 | 0.537 | 0.532 | 0.534 |
| | Conformer (16 Channel) | 77.06 | **0.584** | **0.574** | **0.584** | **0.578** |

Table 4.II – Model results on Pilot Dataset. Asterisk (*) indicates a macro averaging for the given value.

Although there was a decrease in F1 score, these results highlight the existence of electrical brain activity trends that could foreshadow a decrease in behavioral performance. The flanker dataset that was used was quite small in contrast to common transformer training datasets, yet the transformer was still able to receive an adequate score. With an increase in training data, we believe that there could be a substantial increase in classifier performance. Furthermore, our polling began 1 second prior to the stimulus due to constraints in Flanker testing procedures. We also believe that increasing the polling band prior to the stimulus will also increase the classifier performance, and transferability. The transformer model works exceptionally well in real time, which would be its main use case when dealing with BCI devices. It has an inference speed of 0.01 seconds, resulting in the bulk of computation time being associated with input processing.

# CHAPTER 5

# CONCLUSION

## 5.1 Deviation Detection

Based on the results, we conclude that it is certainly possible to perform real time deviation assessments of piloting tasks. We can see that it is also possible to combine both data- and knowledge-driven process-mining methods to capture an accurate representation of a correct process execution. The knowledge-driven models provided by the Aircraft Ontology serves as a structural backbone for this deviation model, though with certain tasks required to be performed in tandem, a strict assessment cannot be directly performed. By utilizing the abilities of the Siamese LSTM, we can perform better assessments of correctly executed task sequences by leveraging previously accepted takeoff sequences. Through the synthetic generation of mutations on extracted Aircraft Ontology sequences, we have validated our first hypothesis by assessing deviations of different varieties, including added, omitted, and out of sequence errors, while also understanding relationships between tasks. These relational tasks will not be flagged if deemed in conformance by the Siamese LSTM.

## 5.2 Pre-Error State Detection

With such small margins for error in real-time systems, such as aviation, the need for predictive error detention is crucial. In this study, we have demonstrated the ability of different machine learning models to predict erroneous states within a perceptual-decision-making flanker and piloting task, using EEG. These signals were recorded prior to the behavioral response and achieved a maximum macro averaged F1 score of 0.601 with a transformer-based model. Furthermore, testing the transferability of this model to an aircraft piloting task yielded promising results with a maximum F1 score of 0.578, suggesting that the signals in behavioral responses may be similar across certain domains and can be used to substitute the gaps in data required – consequently accelerating the

creation of error prevention systems that can revolutionize transportation safety. It is important to address the limitations of the pre-error detection procedure, particularly concerning the use of the Flanker test to train our machine learning models. The Flanker test is relatively simple and may not fully capture the complexity and variety of errors that airplane pilots encounter in real-world scenarios. While the Flanker test provides a controlled environment to identify pre-error signals in the anterior cingulate cortex (ACC), it does not encompass the multifaceted cognitive demands and stressors present during actual piloting tasks. While this study highlights the potential to predict errors, there is room for improvement with further research, more complex experimentation, and larger datasets.

## 5.3 Future Work

The findings presented in this thesis open avenues for future research to further enhance the capabilities of deviation detection and pre-error state detection systems in critical domains such as aviation. Several areas warrant exploration and refinement to improve the effectiveness and applicability of these systems:

### 5.3.1 Refinement of Machine Learning Models

While the results demonstrate promising performance of machine learning models in predicting erroneous states, further refinement and optimization of these models are necessary. Future research efforts should explore advanced techniques, such as ensemble learning, attention mechanisms, or more expansive datasets, to improve predictive accuracy and robustness across different task domains.

### 5.3.2 Exploration of Multimodal Data Fusion

Integrating multiple data modalities, such as eye tracking, and pulse, can provide richer insights into cognitive processes and behavioral responses. Future studies could investigate techniques for multimodal data fusion to enhance the discriminative power

of error prediction models and uncover underlying patterns that may not be evident from individual data sources alone.

### 5.3.3 Optimization of Signal Processing Techniques

Incorporating signal processing techniques such as low-pass filters with a cutoff frequency of 50 Hz and downsampling at 100 Hz, as suggested by the Nyquist-Shannon sampling theorem, could improve the efficiency and information retention of the data preprocessing pipeline. While the use of a notch filter at 50 Hz was based on initial readings, further experimentation with additional signal processing methods could yield performance improvements without sacrificing critical information. Additionally, exploring the balance between computational performance and the amount of information fed into machine learning models could lead to optimized data representations for enhanced predictive capabilities.

# BIBLIOGRAPHY


[1] No survivors as pakistan plane crash kills 152. https://www.bbc.com/news/world-10784971, Jul 2010. Accessed: 2024-03-13.

[2] Airasia crash: crew lost control of plane after apparent misunderstanding. https://www.theguardian.com/world/2015/dec/01/airasia-crew-actions-caused-jet-to-lose-control-say-crash-investigators, Dec 2015. Accessed: 2024-03-13.

[3] Levers of power: The crash of yeti airlines flight 691. https://admiralcloudberg.medium.com/levers-of-power-the-crash-of-yeti-airlines-flight-691-caedd8f8f7e0, Jan 2024. Accessed: 2024-03-13.

[4] Robert L Helmreich. On error management: lessons from aviation. *BMJ*, 320(7237):781–785, 2000.

[5] A. Mehmood, M. Maqsood, M. Bashir, and Y. Shuyuan. A Deep Siamese Convolution Neural Network for Multi-Class Classification of Alzheimer Disease. *Brain Sci*, 10(2), Feb 2020.

[6] What is fdm in aviation? (flight data monitoring). https://termaviation.com/what-is-fdm-in-aviation/, September 2023. Accessed: 2024-04-17.

[7] What is a flight data monitoring programme? https://www.easa.europa.eu/community/topics/flight-data-monitoring, January 2022. Accessed: 2024-04-17.

[8] what is flight data monitoring? https://swiss49.com/fundamentals/what-is-flight-data-monitoring/, November 2023. Accessed: 2024-04-17.

[9] Flight data monitoring (fdm). https://skybrary.aero/articles/flight-data-monitoring-fdm, April 2024. Accessed: 2024-04-17.

[10] Advances in flight data monitoring. https://maxcraft.ca/wp-content/uploads/2012/01/2013-Advances-in-Flight-Data-Montoring-JanFeb.pdf, January 2013. Accessed: 2024-04-17.

[11] Julian Oehling and David J. Barry. Using machine learning methods in airline flight data monitoring to generate new operational safety knowledge from existing data. *Safety Science*, 114:89–104, April 2019.

[12] Artificial intelligence (ai) vs. machine learning (ml). https://cloud.google.com/learn/artificial-intelligence-vs-machine-learning, March 2024. Accessed: 2024-04-17.

[13] Machine learning vs. ai: Differences, uses, and benefits. https://www.coursera.org/articles/machine-learning-vs-ai, March 2024. Accessed: 2024-04-17.

[14] Machine learning in finance: 10 applications and use cases. https://www.coursera.org/articles/machine-learning-in-finance, February 2024. Accessed: 2024-04-17.

[15] Machine learning (in finance). https://corporatefinanceinstitute.com/resources/data-science/machine-learning-in-finance/, February 2024. Accessed: 2024-04-17.

[16] 16 machine learning in healthcare examples). https://builtin.com/artificial-intelligence/machine-learning-healthcare, March 2024. Accessed: 2024-04-18.

[17] What is machine learning in healthcare? applications and opportunities. https://www.coursera.org/in/articles/machine-learning-in-health-care, November 2023. Accessed: 2024-04-18.

[18] Transforming patient care: The role of machine learning in healthcare. https://www.thoughtful.ai/blog/transforming-patient-care-the-role-of-machine-learning-in-healthcare, March 2024. Accessed: 2024-04-18.

[19] Machine learning and marketing: Tools, examples, and tips most teams can use. https://blog.hubspot.com/marketing/machine-learning-and-marketing, October 2023. Accessed: 2024-04-17.

[20] Introduction to machine learning for marketing. https://www.forbes.com/sites/theyec/2022/11/08/introduction-to-machine-learning-for-marketing/?sh=137d45e017b0, November 2022. Accessed: 2024-04-17.

[21] Machine learning for transportation. https://mobility.mit.edu/machine-learning, November 2023. Accessed: 2024-04-17.

[22] David Mhlanga. *Artificial Intelligence and Machine Learning in Making Transport, Safer, Cleaner, More Reliable, and Efficient in Emerging Markets*, page 193–211. Springer Nature Switzerland, 2023.

[23] Navigating the skies with machine learning: Predicting the future in aviation. https://www.aviationfile.com/machine-learning-in-aviation/, November 2023. Accessed: 2024-04-17.

[24] Ai in aviation and airlines: Use cases for 2024. https://mindtitan.com/resources/industry-use-cases/ai-in-aviation-and-travel/, March 2023. Accessed: 2024-04-17.

[25] Ml with large datasets. https://trymachinelearning.com/ml-with-large-datasets/, November 2017. Accessed: 2024-04-12.

[26] What is a machine learning algorithm? https://www.ibm.com/topics/machine-learning-algorithms, March 2024. Accessed: 2024-04-12.

[27] Supervised machine learning. https://www.geeksforgeeks.org/supervised-machine-learning/, February 2024. Accessed: 2024-04-14.

[28] What is unsupervised learning? https://cloud.google.com/discover/what-is-unsupervised-learning, March 2024. Accessed: 2024-04-14.

[29] What is semi-supervised learning? https://www.ibm.com/topics/semi-supervised-learning, March 2024. Accessed: 2024-04-17.

[30] What is reinforcement learning? https://www.ibm.com/topics/reinforcement-learning, March 2024. Accessed: 2024-04-17.

[31] What is deep learning? https://www.mathworks.com/discovery/deep-learning.html, November 2023. Accessed: 2024-04-17.

[32] What is transfer learning? https://www.ibm.com/topics/transfer-learning, March 2024. Accessed: 2024-04-17.

[33] Generative adversarial network. https://deepai.org/machine-learning-glossary-and-terms/generative-adversarial-network, June 2022. Accessed: 2024-04-17.

[34] Sindhu V. An empirical science research on bioinformatics in machine learning. *JOURNAL OF MECHANICS OF CONTINUA AND MATHEMATICAL SCIENCES*, spl7(1), February 2020.

[35] What are support vector machines (svms)? https://www.ibm.com/topics/support-vector-machine, March 2024. Accessed: 2024-04-17.

[36] Tao Liu, P. R. Kumar, Ruida Zhou, and Xi Liu. Learning from few samples: Transformation-invariant svms with composition and locality at multiple scales, 2021.

[37] Vladimir N. Vapnik. *The Support Vector method*, page 261–271. Springer Berlin Heidelberg, 1997.

[38] Random forests. https://deepai.org/machine-learning-glossary-and-terms/random-forest, March 2024. Accessed: 2024-04-17.

[39] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

[40] Feature importances with a forest of trees. https://scikit-learn.org/stable/auto$_e$*xamples*/*ensemble*/*plot*$_f$*orest*$_i$*mportances.html*,*March*2024.*Accessed* : 2024 – 04 – 17.

[41] Random decision forests in finance: Preparing for the unexpected. https://www.bairesdev.com/blog/random-decision-forests-in-finance/, March 2024. Accessed: 2024-04-17.

[42] Shannon Wongvibulsin, Katherine C. Wu, and Scott L. Zeger. Clinical risk prediction with random forests for survival, longitudinal, and multivariate (rf-slam) data analysis. *BMC Medical Research Methodology*, 20(1), December 2019.

[43] Yanjun Qi. *Random Forest for Bioinformatics*, page 307–323. Springer New York, 2012.

[44] Autoencoders -machine learning. https://www.geeksforgeeks.org/auto-encoders/, December 2023. Accessed: 2024-04-17.

[45] Denoising autoencoders. https://deepai.org/machine-learning-glossary-and-terms/denoising-autoencoder, March 2024. Accessed: 2024-04-17.

[46] Dimensionality reduction using deep learning: Autoencoder. https://socr.umich.edu/HTML5/ABIDE$_A$*utoencoder*/,*March*2024. *Accessed* : 2024 – 04 – 17.

[47] Lucas Cazzonelli and Cedric Kulbach. *Detecting Anomalies with Autoencoders on Data Streams*, page 258–274. Springer International Publishing, 2023.

[48] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[49] Yingjie Zhang, Hong Geok Soon, Dongsen Ye, Jerry Ying Hsi Fuh, and Kunpeng Zhu. Powder-bed fusion process monitoring by machine vision with hybrid convolutional neural networks. *IEEE Transactions on Industrial Informatics*, 16(9):5769–5779, September 2020.

[50] Image classification using cnn. https://datagen.tech/guides/image-classification/image-classification-using-cnn/, March 2024. Accessed: 2024-04-17.

[51] Introduction to convolutional neural networks (cnn). https://www.educative.io/blog/introduction-to-convolutional-neural-networks, March 2024. Accessed: 2024-04-17.

[52] Sunitha Basodi, Chunyan Ji, Haiping Zhang, and Yi Pan. Gradient amplification: An efficient way to train deep neural networks. *Big Data Mining and Analytics*, 3(3):196–207, 2020.

[53] Exploring the lstm neural network model for time series. https://towardsdatascience.com/exploring-the-lstm-neural-network-model-for-time-series-8b7685aa8cf, January 2022. Accessed: 2024-04-17.

[54] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 12(10):2451–2471, October 2000.

[55] Language translation with rnns. https://towardsdatascience.com/language-translation-with-rnns-d84d43b40571, February 2019. Accessed: 2024-04-17.

[56] Sentiment analysis — using lstm glove embeddings. https://medium.com/@skillcate/sentiment-classification-using-neural-networks-a-complete-guide-1798aaf357cd, July 2022. Accessed: 2024-04-17.

[57] Long short-term memory networks (lstm). https://databasecamp.de/en/ml/lstms, June 2022. Accessed: 2024-04-17.

[58] A deep dive into the transformer architecture — the development of transformer models. https://towardsdatascience.com/a-deep-dive-into-the-transformer-architecture-the-development-of-transformer-models-acbdf7ca34e0, March 2024. Accessed: 2024-04-17.

[59] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[60] Sabeen Ahmed, Ian E. Nielsen, Aakash Tripathi, Shamoon Siddiqui, Ghulam Rasool, and Ravi P. Ramachandran. Transformers in time-series analysis: A tutorial. 2022.

[61] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation, 2015.

[62] Dynamic programming or dp. https://www.geeksforgeeks.org/dynamic-programming/, April 2024. Accessed: 2024-04-17.

[63] What is dijkstra's algorithm? | introduction to dijkstra's shortest path algorithm. https://www.geeksforgeeks.org/introduction-to-dijkstras-shortest-path-algorithm/, March 2024. Accessed: 2024-04-17.

[64] Kunping Yang, Gui-Song Xia, Zicheng Liu, Bo Du, Wen Yang, Marcello Pelillo, and Liangpei Zhang. Asymmetric siamese networks for semantic change detection in aerial images. *IEEE Transactions on Geoscience and Remote Sensing*, 60:1–18, 2022.

[65] Luis Basora, Xavier Olive, and Thomas Dubot. Recent advances in anomaly detection methods applied to aviation. *Aerospace*, 6(11), 2019.

[66] Florian Frische, Tomasz Mistrzyk, and Andreas Lüdtke. Detection of pilot errors in data by combining task modeling and model checking. In Tom Gross, Jan Gulliksen, Paula Kotzé, Lars Oestreicher, Philippe Palanque, Raquel Oliveira Prates, and Marco Winckler, editors, *Human-Computer Interaction – INTERACT 2009*, pages 528–531, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[67] Sen Yang, Aleksandra Sarcevic, Richard A. Farneth, Shuhong Chen, Omar Z. Ahmed, Ivan Marsic, and Randall S. Burd. An approach to automatic process deviation detection in a time-critical clinical process. *Journal of Biomedical Informatics*, 85:155–167, 2018.

[68] S. C. Christov, G. S. Avrunin, and L. A. Clarke. Online deviation detection for medical processes. *AMIA Annu Symp Proc*, 2014:395–404, 2014.

[69] Flávio L. Lázaro, Rui P. R. Nogueira, Rui Melicio, Duarte Valério, and Luís F. F. M. Santos. Human factors as predictor of fatalities in aviation accidents: A neural network analysis. *Applied Sciences*, 14(2):640, January 2024.

[70] Rui P. R. Nogueira, Rui Melicio, Duarte Valério, and Luís F. F. M. Santos. Learning methods and predictive modeling to identify failure by human factors in the aviation industry. *Applied Sciences*, 13(6):4069, March 2023.

[71] Mohammed Badawy, Nagy Ramadan, and Hesham Ahmed Hefny. Healthcare predictive analytics using machine learning and deep learning techniques: a survey. *Journal of Electrical Systems and Information Technology*, 10(1), August 2023.

[72] Nabila Sghir, Amina Adadi, and Mohammed Lahmer. Recent advances in predictive learning analytics: A decade systematic review (2012–2022). *Education and Information Technologies*, 28(7):8299–8333, December 2022.

[73] Janis Peksa and Dmytro Mamchur. State-of-the-art on brain-computer interface technology. *Sensors*, 23(13):6001, June 2023.

[74] Baraka Maiseli, Abdi T. Abdalla, Libe V. Massawe, Mercy Mbise, Khadija Mkocha, Nassor Ally Nassor, Moses Ismail, James Michael, and Samwel Kimambo. Brain–computer interface: trend, challenges, and threats. *Brain Informatics*, 10(1), August 2023.

[75] Kuan-Jung Chiang, Dimitra Emmanouilidou, Hannes Gamper, David Johnston, Mihai Jalobeanu, Edward Cutrell, Andrew Wilson, Winko W. An, and Ivan Tashev. A closed-loop adaptive brain-computer interface framework: Improving the classifier with the use of error-related potentials. In *2021 10th International IEEE/EMBS Conference on Neural Engineering (NER)*, pages 487–490, 2021.

[76] Maren David Dangut, Ian K. Jennions, Steve King, and Zakwan Skaf. A rare failure detection model for aircraft predictive maintenance using a deep hybrid learning approach. *Neural Computing and Applications*, 35(4):2991–3009, March 2022.

[77] Khalid Khan, Muhammad Sohaib, Azaz Rashid, Saddam Ali, Hammad Akbar, Abdul Basit, and Tanvir Ahmad. Recent trends and challenges in predictive maintenance of aircraft's engine and hydraulic system. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 43(8), August 2021.

[78] Alexander E. Hramov, Vladimir A. Maksimenko, and Alexander N. Pisarchik. Physical principles of brain–computer interfaces and their applications for rehabilitation, robotics and control of human brain states. *Physics Reports*, 918:1–133, 2021. Physical principles of brain–computer interfaces and their applications for rehabilitation, robotics and control of human brain states.

[79] Michael L Martini, Eric Karl Oermann, Nicholas L Opie, Fedor Panov, Thomas Oxley, and Kurt Yaeger. Sensor modalities for brain-computer interface technology: A comprehensive literature review. *Neurosurgery*, 86(2):E108–E117, July 2019.

[80] T Koenig, D Studer, D Hubl, L Melie, and W.K Strik. Brain connectivity at different time-scales measured with eeg. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 360(1457):1015–1024, May 2005.

[81] Cameron S. Carter, Todd S. Braver, Deanna M. Barch, Matthew M. Botvinick, Douglas Noll, and Jonathan D. Cohen. Anterior cingulate cortex, error detection, and the online monitoring of performance. *Science*, 280(5364):747–749, May 1998.

[82] Clay B. Holroyd and Michael G. H. Coles. The neural basis of human error processing: Reinforcement learning, dopamine, and the error-related negativity. *Psychological Review*, 109(4):679–709, October 2002.

[83] K.Richard Ridderinkhof, Sander Nieuwenhuis, and Theodore R. Bashore. Errors are foreshadowed in brain potentials associated with action monitoring in cingulate cortex in humans. *Neuroscience Letters*, 348(1):1–4, September 2003.

[84] Alisa Batmanova, Alexander Kuc, Vladimir Maksimenko, Andrey Savosenkov, Nikita Grigorev, Susanna Gordleeva, Victor Kazantsev, Sergey Korchagin, and Alexander

Hramov. Predicting perceptual decision-making errors using eeg and machine learning. *Mathematics*, 10(17):3153, September 2022.

[85] Rui Li, Le-Dian Liu, and Bao-Liang Lu. Discrimination of decision confidence levels from eeg signals. In *2021 10th International IEEE/EMBS Conference on Neural Engineering (NER)*. IEEE, May 2021.

[86] H Zhang, R Chavarriaga, Z Khaliliardali, L Gheorghe, I Iturrate, and J d R Millán. Eeg-based decoding of error-related brain activity in a real-world driving task. *Journal of Neural Engineering*, 12(6):066028, November 2015.

[87] Maxime Antoine, Hamdi Ben Abdessalem, and Claude Frasson. Cognitive workload assessment of aircraft pilots. *Journal of Behavioral and Brain Science*, 12(10):474–484, 2022.

[88] Robert Oostenveld and Peter Praamstra. The five percent electrode system for high-resolution eeg and erp measurements. *Clinical Neurophysiology*, 112(4):713–719, April 2001.

[89] *Electroencephalography and Clinical Neurophysiology*, 10(2):370–375, May 1958.

[90] Marc-Antoine Courtemanche, Ange Tato, and Roger Nkambou. Ontological reference model for piloting procedures. In Scott Crossley and Elvira Popescu, editors, *Intelligent Tutoring Systems*, pages 95–104, Cham, 2022. Springer International Publishing.

[91] Web ontology language (owl). https://www.w3.org/OWL/. Accessed: 2022-09-30.

[92] Zeromq: An open-source universal messaging library. https://zeromq.org/, March 2024. Accessed: 2024-04-18.

[93] J. Chao, F. Tang, and L. Xu. Developments in Algorithms for Sequence Alignment: A Review. *Biomolecules*, 12(4), Apr 2022.

[94] J. Rose and F. Eisenmenger. A fast unbiased comparison of protein structures by means of the Needleman-Wunsch algorithm. *J Mol Evol*, 32(4):340–354, Apr 1991.

[95] An Na. Anomalies detection and tracking using siamese neural networks. Master's thesis, Auckland University of Technology, 2020.

[96] Anfeng He, Chong Luo, Xinmei Tian, and Wenjun Zeng. A twofold siamese network for real-time object tracking, 2018.

[97] Gregory Koch. Siamese neural networks for one-shot image recognition. Master's thesis, University of Toronto, 2015.

[98] Marcel F. Hinss, Emilie S. Jahanpour, Bertille Somon, Lou Pluchon, Frédéric Dehais, and Raphaëlle N. Roy. Open multi-session and multi-task eeg cognitive dataset for passive brain-computer interface applications. *Scientific Data*, 10(1), February 2023.

[99] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, June 2002.

[100] Alberto Fernández, Salvador García, Mikel Galar, Ronaldo Prati, Bartosz Krawczyk, and Francisco Herrera. *Foundations on Imbalanced Classification*, pages 19–46. 01 2018.

[101] B. Richhariya and M. Tanveer. Eeg signal classification using universum support vector machine. *Expert Systems with Applications*, 106:169–182, September 2018.

[102] Damodar Reddy Edla, Kunal Mangalorekar, Gauri Dhavalikar, and Shubham Dodia. Classification of eeg data for human mental state analysis using random forest classifier. *Procedia Computer Science*, 132:1523–1532, 2018.

[103] Clay B Holroyd and Michael G H Coles. The neural basis of human error processing: reinforcement learning, dopamine, and the error-related negativity. *Psychol. Rev.*, 109(4):679–709, October 2002.

[104] Stephan F. Taylor, Emily R. Stern, and William J. Gehring. Neural systems for error monitoring: Recent findings and theoretical perspectives. *The Neuroscientist*, 13(2):160–172, April 2007.

[105] Massimo Pietracupa, Hamdi Ben Abdessalem, and Claude Frasson. An approach to automatic flight deviation detection. In Claude Frasson, Phivos Mylonas, and Christos Troussas, editors, *Augmented Intelligence and Intelligent Tutoring Systems*, pages 530–540, Cham, 2023. Springer Nature Switzerland.

[106] Juri Opitz and Sebastian Burst. Macro f1 and macro f1, 2021.

[107] Massimo Pietracupa, Hamdi Ben Abdessalem, and Claude Frasson. Detection of pre-error states in aircraft pilots through machine learning. 2024. forthcoming.