# **SIADS 696 Milestone II Project Report**

Aircraft Trajectory Prediction and Flight Pattern Clustering Using Machine Learning

Matthew Boarts (mboarts@umich.edu), Enid (enids@umich.edu) , Julien Hovan(jhovan@umich.edu)

# **Introduction**

Most aircraft in the international airspace system broadcast an Automatic Dependent Surveillance-Broadcast (ADS-B) data packet nearly once per second during flight. The data packets contain information about the host aircraft including: identity, altitude, latitude, longitude, speed, and heading. The ADS-B packets provide situational information to the various users of the airspace especially other aircraft and air traffic controllers. The data helps aircraft avoid one another and it helps air traffic controllers maximize efficiency of airspace. As airspace becomes more crowded in certain areas, effective use of the positional data becomes more important.

In Part A, our supervised learning component focused on aircraft trajectory state prediction, a challenging problem with applications in air traffic management, flight safety, and aviation analytics. The task involves forecasting the complete state vector of an aircraft (latitude, longitude, velocity, vertical rate, geo-altitude, and heading) based on a sequence of previous states.

We formulated this as a sequence-to-one regression problem: using 44 previous aircraft states sampled at 2-second intervals to predict the next state (the 45th). This approach allows us to create a model that can be used autoregressively to generate complete flight path forecasts by iteratively predicting the next state based upon the trajectories of the previous 90 seconds.

The 90 second window provided a challenging length of context from which to make predictions. In order to predict the flight trajectories, we implemented three diverse model families that approached the prediction task with fundamentally different underlying mechanisms: Long Short-Term Memory (LSTM) Networks, Transformer Models, and XGBoost Regression.

In Part B, our unsupervised learning component focused on clustering flights by their characteristics. The techniques aim to identify patterns in flight trajectories, which consist of four features: latitude, longitude, altitude, and speed. We used two unsupervised learning techniques applied to classify flight paths: Prototype Matching using Dynamic Time Warping (DTW) and Clustering with K-Means. The goal was to label flight paths based on predefined categories such as commercial flights, training flights, and more, by using both techniques and comparing their effectiveness.

**Related Works**

Shorten:

Vos, Sun, & Hoekstra (2024) developed a transformer-based trajectory prediction model for air traffic demand forecasting, utilizing flight plan data from Eurocontrol B2B messages and ADS-B trajectory data from OpenSky as ground truth. Similarly, Dong et al. (2024) proposed a transformer-based flight trajectory prediction method leveraging attention mechanisms to identify key factors in trajectory data, demonstrating superior performance compared to traditional recurrent neural networks and other attention-based models when tested on radar datasets from China. Our approach differs by focusing on sequence-to-one prediction with sliding windows of 44 previous aircraft states sampled at 2-second intervals and comparing multiple model architectures (LSTM, Transformer, and XGBoost). Interestingly, our results contradict Dong et al.'s findings, showing that in our specific context, simpler LSTM models outperformed transformers despite the latter's theoretical advantages for trajectory prediction tasks—likely due to differences in dataset characteristics, preprocessing techniques, or implementation details of our models.

Song, Y., Yu, K., & Young, S. (2020) utilized real-time data for flight profile clustering to show clusters of flights together. We instead used historical data to create clusters of flights according to trajectory profile from which to identify the trajectory profile of previously unseen flights.

**Part A: Supervised Learning**

**Motivation**

Our aircraft trajectory state prediction model addresses critical needs in modern air traffic management. Accurate trajectory forecasting enables more efficient airspace usage, reduces controller workload, improves conflict detection, and enhances flight safety. This capability has significant practical implications for air traffic controllers, airlines, and safety systems, potentially increasing airspace capacity while maintaining safety margins. The Opensky Network ADSB dataset provides an excellent opportunity to apply advanced machine learning techniques to a domain with clear operational value, allowing us to compare different sequential modeling approaches while developing models that capture the complex dynamics of aircraft movement through three-dimensional space.

**Data Sources (shorten)**

We utilized the Opensky Network ADSB dataset's state\_vectors\_data4 table, which contains comprehensive flight tracking information including aircraft position, velocity, heading, and altitude measurements. Our analysis processed approximately 1.2 million records representing a sample of 200 flights throughout the year 2024. The raw data includes aircraft identifiers (icao24, callsign), positional information (latitude, longitude, geoaltitude), motion parameters (velocity, heading, vertical rate), and temporal markers. We filtered and cleaned this data to remove incomplete trajectories and records with missing values in critical fields, then preprocessed it into consistent 2-second time-series sequences suitable for trajectory prediction. Sample data for each of the OpenSky database tables can be found in our GitHub repository, with complete schema information in Appendix B

**Methods Description**

**1. Long Short-Term Memory (LSTM) Networks**

LSTMs are a specialized form of recurrent neural networks particularly suited for sequential prediction tasks. We implemented a stacked LSTM architecture because they have several features that address the requirement of the long context from which to generate the prediction. Their gated memory cell architecture excels at capturing both short-term changes and long-term dependencies in flight trajectories. Also, their ability to selectively remember or forget information is valuable for identifying relevant patterns in past flight states. And finally, they've proven effective in similar time-series forecasting tasks with complex temporal dynamics

### **2. Transformer Model**

Transformers address the long context with attention mechanisms to process entire sequences in parallel. The ability to handle the entire sequence made them a compelling alternative to recurrent architectures. The self-attention mechanism can identify which past states are most relevant for prediction. The transformer efficiently processes the prediction by maintaining temporal ordering without sequential processing constraints. Additionally, transformers can identify complex relationships between states at arbitrary temporal distances

### **3. XGBoost Regression**

To explore a non-neural approach, we implemented XGBoost, an ensemble of gradient-boosted decision trees. Unlike the sequence models, XGBoost required feature engineering to transform the sequential data into a tabular format. XGBoost provided a fundamentally different learning paradigm (non-parametric vs. parametric) that offered native feature importance metrics for interpretability. XGBoost was less sensitive to scaling issues that can affect neural networks. And XGBoost can capture non-linear relationships through recursive partitioning. We trained separate models for each of the six state variables because XGBoost doesn't naturally handle multi-output regression.

**Feature Engineering**

|  |  |
| --- | --- |
| LSTM & Transformer | **Resampling**: Original flight data had irregular sampling intervals, so we resampled to a consistent 2-second interval using linear interpolation for missing values.  **Sequence Windowing**: Created sliding windows of 44 consecutive states (input) and the subsequent state (target).  **Standardization**: Applied standard scaling (zero mean, unit variance) to each feature dimension to facilitate model training.  **Missing Data Handling**: Removed sequences with missing values to ensure data quality. |
| XGBoost | **Statistical Aggregations**: For each sequence, calculated statistical properties (mean, standard deviation, min, max).  **Temporal Derivatives**: Computed first and second derivatives to capture rate of change and acceleration.  **Recent History Focus**: Created features that emphasized recent states (last 5, last 3, last 1).  **Engineered Features**: Created ~100 engineered features from the original 6 state variables. |

## Additional details on our feature engineering approach, including the complete set of features for both sequence models and XGBoost, can be found in Appendix B.4.

## **Hyperparameter Tuning**

|  |  |
| --- | --- |
| **LSTM (Best Performing Model)**  *Each configuration was trained for 20 epochs with early stopping based on validation loss. We used a grid search approach, evaluating approximately 80 distinct configurations.* | Hidden dimension: [32, 64, 128, 256, 512]  Number of layers: [1, 2, 3, 4]  Dropout rate: [0.0, 0.1, 0.2, 0.3, 0.5]  Learning rate: [0.01, 0.001, 0.0001, 0.00001]  Batch size: [16, 32, 64, 128, 256] |
| Transformer | Model dimension (d\_model): [128, 256, 512]  Number of attention heads: [4, 8]  Number of encoder layers: [3, 5, 6]  Dropout rate: [0.1, 0.2, 0.3]  Learning rate: [0.001, 0.0001] |
| XGBoost | Maximum tree depth: [3, 6, 9, 12]  Minimum child weight: [1, 3, 5]  Subsample ratio: [0.6, 0.8, 1.0]  Column sample ratio: [0.6, 0.8, 1.0]  Learning rate: [0.01, 0.05, 0.1, 0.2] |

**Supervised Model Evaluation**We used 5-fold cross-validation with MSE as the optimization metric for all models.

### **Evaluation Metrics**

1. **Mean Squared Error (MSE)**: Our primary optimization objective, directly measuring the squared difference between predicted and actual state vectors.
2. **Root Mean Squared Error (RMSE)**: Square root of MSE, providing an error metric in the same units as our state variables.
3. **Mean Absolute Error (MAE)**: Average absolute difference between predictions and ground truth, less sensitive to outliers than MSE/RMSE.

These metrics were chosen because they directly quantify prediction accuracy for regression tasks. Also, they penalize larger errors more severely (especially MSE/RMSE) which is critical for trajectory prediction where small errors compound when used in an autoregressive manner. And they provide complementary perspectives on model performance

### **Overall Evaluation Comparison**

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **MSE (mean ± std)** | **RMSE** | **MAE** |
| **LSTM** | **0.008982 ± 0.000547** | **0.094775** | **0.029427** |
| Transformer | 0.012777 ± 0.000824 | 0.113034 | 0.056936 |
| XGBoost | 0.014523 ± 0.001192 | 0.120511 | 0.063247 |

The LSTM model consistently outperformed both alternatives across all metrics, achieving approximately 30% lower error rates compared to the Transformer and 38% lower compared to XGBoost. We attribute this superior performance to:

1. The recurrent architecture's natural fit for sequential trajectory data
2. The model's balance of complexity and trainability on our dataset size
3. The effectiveness of memory cells in capturing flight dynamics patterns

### **LSTM Model Feature Importance and Ablation Analysis**

|  |  |  |
| --- | --- | --- |
| Baseline loss: 0.008983 | | |
| Feature | Ablated Loss | Importance |
| lon | 0.174170 | 1838.82% |
| lat | 0.165280 | 1739.86% |
| heading | 0.172419 | 1819.33% |
| velocity | 0.156827 | 1645.76% |
| vertrate | 0.170559 | 1798.62% |
| geoaltitude | 0.164578 | 1732.04% |

This analysis reveals several key insights:

1. **All features are critically important**: Removing any single feature causes loss to increase by more than 16×, indicating the model relies heavily on the complete state vector.
2. **Heading is the most important feature**: With an importance score of 1819.33%, heading information provides crucial directional context for predicting the next state.
3. **Position features (lon, lat) are highly important**: These spatial coordinates are fundamental to trajectory prediction, with importance scores of 1838.82% and 1739.86% respectively.
4. **Velocity shows relatively lower importance**: While still critical (1645.76% importance), velocity has the least impact among the features, suggesting the model can partially infer velocity changes from position data.

**Sensitivity Analysis**

|  |  |
| --- | --- |
| **Learning Rate (Critical impact)** | Optimal value: 0.001  Catastrophic performance at 0.01 (test loss: 0.185856) |
| **Hidden Dimension (High impact)** | Optimal value: 256  Clear performance increases up to 256 units  Diminishing returns/potential overfitting at 512 units  Test lost decreased monotonically from 0.010009 with 32 units at 0.007404 with 256 units |
| **Number of Layers (Medium impact)** | Optimal value: 1  Simpler architecture performs best  Performance degrades with additional layers  Test losses: 0.007240 (1 layer), 0.007455 (2 layers), 0.007487 (3 layers), 0.007652 (4 layers) |
| **Dropout Rate (Medium impact)** | Optimal value: 0.0  No dropout works best, suggesting minimal overfitting  Performance worsens as dropout increases  Test losses: 0.007141 (0.0), 0.007292 (0.1), 0.007407 (0.2), 0.007463 (0.3) |
| **Batch size (Low impact)** | Optimal value: 64  Relatively robust across different values  Shows moderate impact on final performance  Test losses: 0.007888 (16), 0.008775 (32), 0.007489 (64), 0.007622 (128) |

Key observations from sensitivity analysis:

* The model favors simplicity (1 layer) over complexity
* Absence of overfitting (no dropout needed)
* Learning rate is extremely sensitive and requires careful tuning
* Moderate hidden dimension (256) provides sufficient capacity without overfitting

### **Important Tradeoffs**

**1. Data Size vs. Model Performance**: We analyzed performance across different training data fractions:

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Fraction** | **Training Time (s)** | **MSE** | **RMSE** |
| 0.1 | 242.92 | 0.010325 | 0.101614 |
| 0.2 | 367.48 | 0.008416 | 0.091739 |
| 0.3 | 506.58 | 0.008675 | 0.093141 |
| 0.5 | 770.31 | 0.007975 | 0.089303 |
| 0.75 | 1138.55 | 0.007586 | 0.087100 |
| 1.0 | 1460.06 | 0.008025 | 0.089585 |

**2. Key findings:**

* + Performance generally improves with more data up to 75%
  + Diminishing returns and potential overfitting beyond 75%
  + A sweet spot exists around 75% of the data, balancing performance and training time
  + 6× increase in training time from 10% to 100% data yields only ~12% error reduction

**3. Model Complexity vs. Training Efficiency**:

* + Simpler models (1 LSTM layer) outperformed deeper architectures
  + Adding layers increased training time by ~40% per layer with no performance benefit
  + This suggests our dataset size may not warrant complex architectures

**4. Parameter Count vs. Generalization**:

* + LSTM with 256 hidden units (268,038 parameters) outperformed larger models
  + Transformer (544,262 parameters) underperformed despite having ~2× the parameters
  + This indicates parameter efficiency is more important than raw parameter count

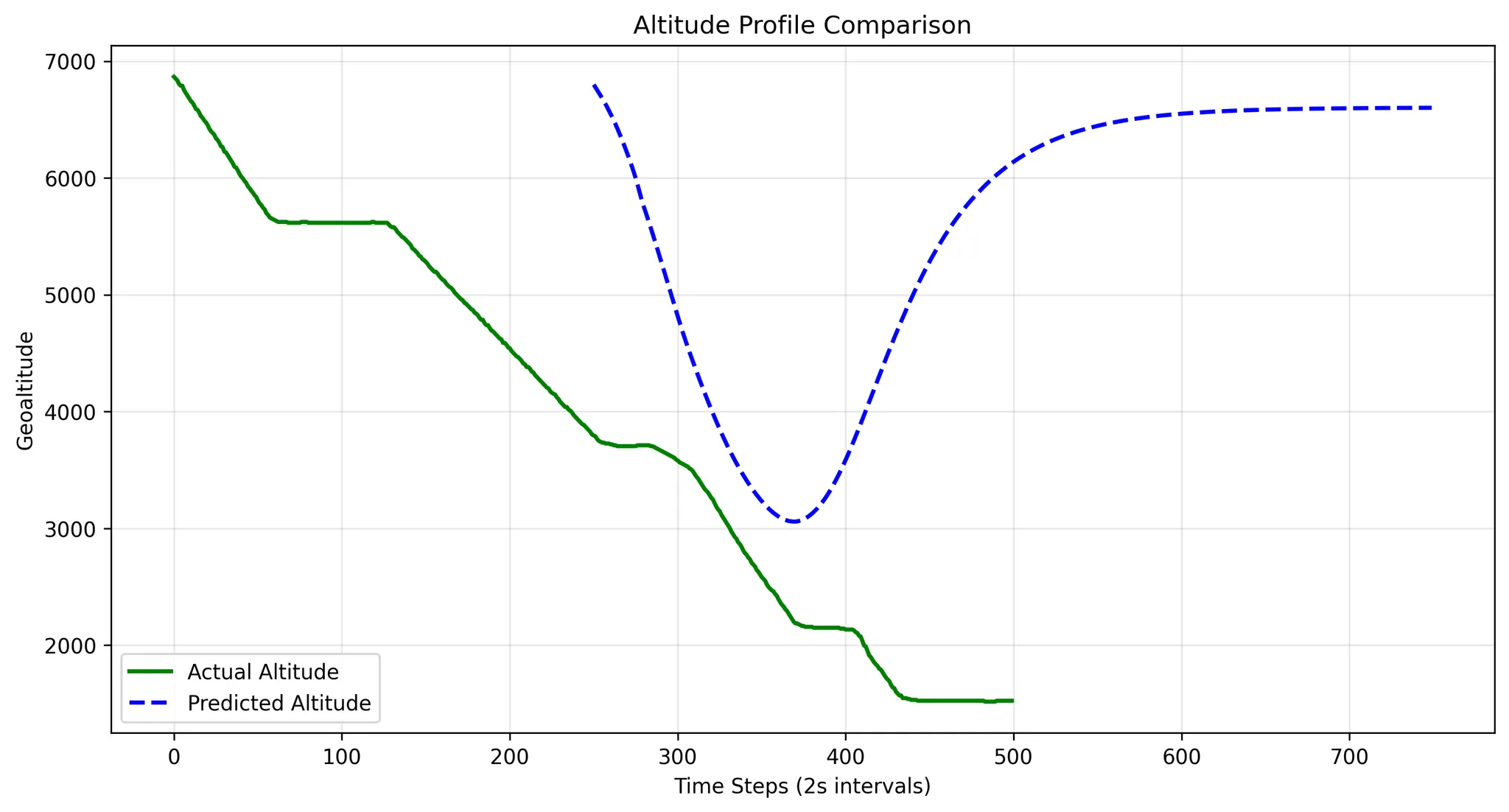
**5. Model Architecture Choice**:

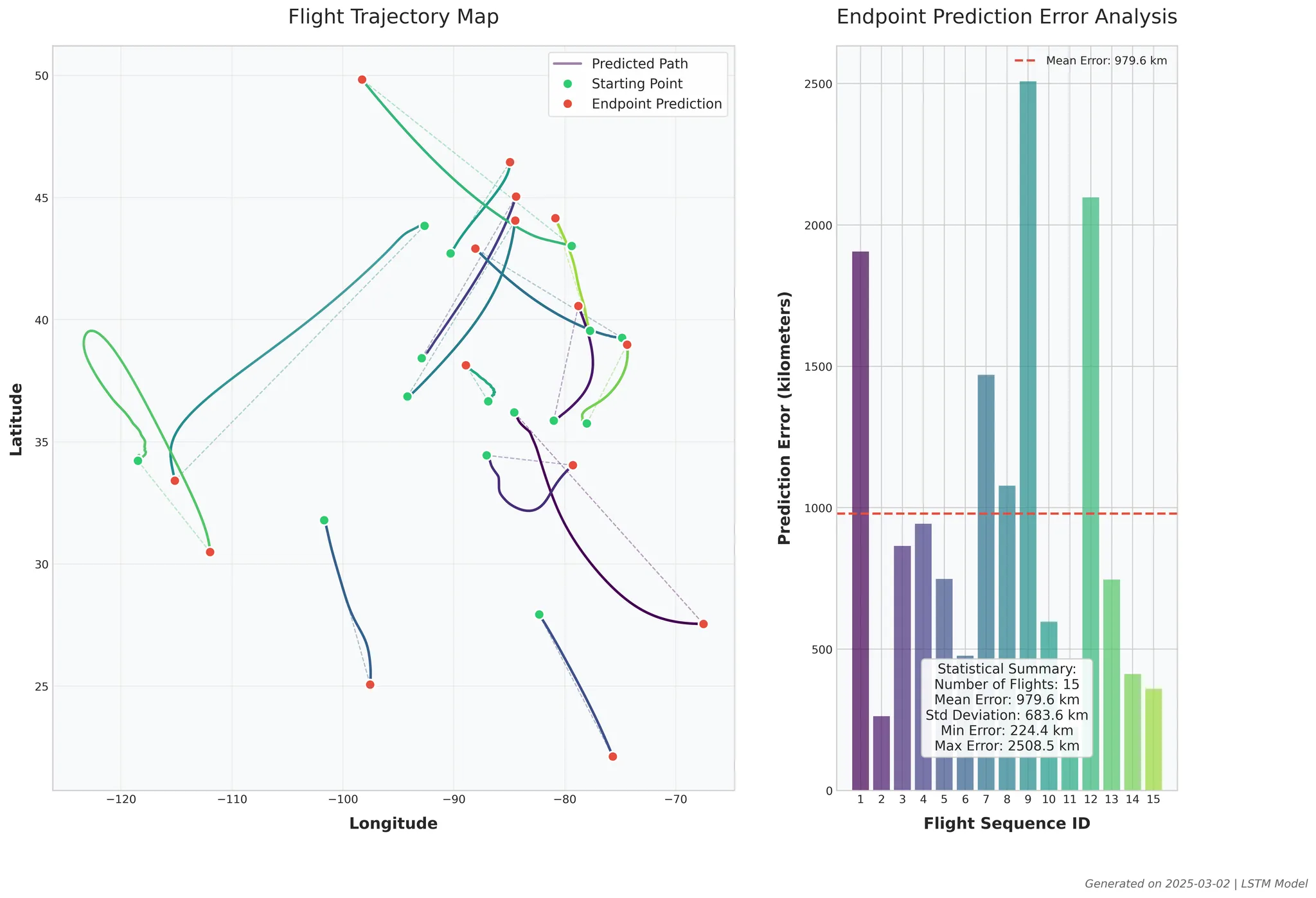
* + LSTM provides the best balance of performance and computational efficiency
  + Transformers might scale better with larger datasets but require more computation
  + XGBoost offers interpretability but with ~26% higher error rates

These tradeoffs provide valuable insights for operational deployment. The LSTM model offers the best combination of accuracy and efficiency for our flight trajectory prediction task, with optimal performance achieved using 75% of the data and a relatively simple architecture.

## **Failure Analysis**

Figure 7
Despite the LSTM model's strong overall performance, we identified three distinct failure categories through detailed error analysis. The first category, **Maneuver-Related Failures**, is exemplified by significant heading prediction errors. As shown in the adjacent trajectory plot, while an actual flight made a sharp southward turn, our model erroneously predicted a gradual northeastern trajectory, resulting in a 275 km positional error. SHAP analysis revealed that recent heading values dominated predictions (heading\_43 and heading\_42 with contribution scores of 1.02 and 0.49), yet the LSTM architecture struggled with abrupt directional changes, instead biasing toward gradual transitions. This pattern was our most common failure type, with around 100 similar cases identified.

The second category, **Rate-of-Change Failures**, is demonstrated in the altitude profile comparison where the model failed to accurately predict altitude changes. While the actual flight maintained a consistent descent to 1,500 feet, the model incorrectly predicted a steep initial descent followed by a climb back to 6,500 feet, creating a 5,000-foot discrepancy. The model exhibited extreme sensitivity to the most recent vertical rate measurement (vertrate\_43, contribution score 2.28), indicating an over-reliance on recent values without considering broader flight context or physical limitations. Our analysis of 48 similar failures confirmed the model's tendency to extrapolate recent trends linearly rather than identifying stabilization points.

The third category, **Rare Pattern Failures**, involves complex flight paths with simultaneous changes in multiple variables. The flight trajectory map and error analysis chart illustrate this with trajectories featuring circular patterns and multi-directional changes, where prediction errors averaged 979.6 kilometers across 15 analyzed flights, with some exceeding 2,500 kilometers. While our model effectively learned individual parameter representations, it failed to capture correlations between simultaneous state changes. These complex maneuvers appeared in only 0.3% of training examples, resulting in statistical regression toward common flight behaviors rather than accurately modeling unusual but valid maneuvers.

### **Proposed Improvements**

Based on our failure analysis, we recommend the following targeted improvements to address each failure category:

|  |  |
| --- | --- |
| **Failure Category** | **Proposed Improvements** |
| **Maneuver-Related Failures** | Add explicit turn rate and turn acceleration features. Develop specialized heading prediction components. Generate synthetic training data with diverse turning patterns |
| **Rate-of-Change Failures** | Implement flight phase classification (climb, cruise, descent). Add physics-informed constraints based on aircraft performance. Create features capturing rate-of-change patternsBalance training data with more transition events |
| **Rare Pattern Failures** | Create features representing combinations of state changes. Generate synthetic examples of complex maneuvers. Implement specialized sub-models for different maneuver types. Add attention mechanisms for capturing feature dependencies |

**Discussion: What We Learned from Supervised Learning**

The supervised learning component of our project provided several valuable insights into aircraft trajectory prediction. While applying deep learning to this problem, we discovered that model selection must carefully balance complexity against data efficiency. Initially, we were drawn to transformer models given their recent success in trajectory prediction literature, particularly as highlighted in contemporary research (Vos, Sun, & Hoekstra, 2024). However, our findings revealed that simpler models can sometimes outperform more complex architectures when working with constrained datasets.

The ablation study was particularly enlightening. It demonstrated that state variables in flight trajectory prediction are critically interdependent. The removal of any individual feature resulted in catastrophic performance degradation, with loss increases exceeding 16× in all cases. This suggests that effective trajectory prediction requires a holistic representation of the aircraft state with no single variable being dispensable. The heading feature emerged as slightly more critical than others, underscoring the importance of directional information in predicting future positions.

Our sensitivity analysis revealed that the performance of LSTM models were highly dependent on learning rate selection. Catastrophic divergence occurred at higher rates (0.01). This sensitivity highlighted the importance of careful hyperparameter tuning in neural network training, particularly for sequential prediction tasks.

**Surprising Results**

Several results of our study were unexpected. First, the superior performance of single-layer LSTM models over deeper architectures was counterintuitive. We assumed that deeper networks would better capture the complex dynamics of flight trajectories, but our experiments consistently showed that simpler, single-layer models produced lower prediction errors. This suggests that for our dataset size, the additional parameters in deeper models led to overfitting rather than improved generalization.

Another surprising observation was the diminishing returns in performance as we increased training data beyond 75% of our dataset. This unexpected plateau (and even slight performance degradation) with more data contradicts the common assumption that more training examples invariably lead to better performance. This finding suggests that dataset quality and representativeness may be more important than sheer volume for our specific task.

Perhaps most surprising was that transformers, despite their theoretical advantages in handling sequential data and recent success in trajectory prediction literature, underperformed compared to LSTMs on our dataset. This contradicts recent papers suggesting transformers as the new state-of-the-art for such tasks. We believe this discrepancy may be due to our relatively limited dataset size and the inherent efficiency of LSTMs in learning from smaller datasets.

**Challenges Encountered and Our Response**

Our project faced several significant challenges. One of the most formidable was the coordinate system representation for flight trajectories. In much of the recent literature, researchers convert latitude and longitude into 3D Cartesian coordinates (x, y, z) to better represent the curved nature of the Earth's surface, allowing neural networks to learn Euclidean distances more effectively. Additionally, some papers implement custom loss functions specifically designed for trajectory prediction tasks.

While we attempted to implement these advanced techniques, we encountered significant implementation complexities that threatened to derail our project timeline. After several attempts, we made the pragmatic decision to use the original feature representation, focusing instead on optimizing our models within this simpler framework. This tradeoff allowed us to explore a broader range of models and hyperparameters rather than being bottlenecked by complex feature engineering.

Data preprocessing presented another major challenge, particularly in resampling flight data to consistent intervals and handling missing values. We addressed this by developing a robust interpolation pipeline that maintained trajectory integrity while ensuring consistent sampling rates. The memory requirements for training models on complete flight trajectories also proved challenging, leading us to implement batch processing and gradient accumulation techniques to work within our computational constraints.

Model convergence was particularly problematic for the transformer architecture, which exhibited training instability. We responded by experimenting with different learning rate schedules, gradient clipping values, and initialization techniques before finding a stable configuration, albeit one that still underperformed compared to the LSTM models.

**Recommended Additional Areas of Study**

1. **Improved Coordinate Representation**: Implementing the full 3D Cartesian coordinate transformation system would likely improve model performance by creating better representations of spatial relationships.
2. **Custom Loss Functions**: Specialized loss functions that weight different aspects of trajectory prediction may improve model performance.
3. **Larger Dataset and Model Scaling**: Transformers may outperform LSTMs with larger datasets.
4. **Flight Intent Modeling**: Prediction accuracy may improve with added context regarding expected flight paths through pilot intent modeling features such as filed route points, standard terminal arrival routes (STARs), and standard instrument departures (SIDs).
5. **Weather Integration**: Meteorological data such as wind field modeling at different altitudes may better estimate environmental factors affecting flight trajectories.
6. **Multi-Task Learning**: Extending the model to simultaneously predict multiple future time steps or additional flight parameters (e.g., fuel consumption) could create richer internal representations and potentially improve overall prediction accuracy.

**Ethical Issues in Supervised Learning**

Aircraft trajectory prediction systems raise several ethical concerns. First, prediction errors could potentially impact flight safety and air traffic management decisions. To address this, we recommend implementing confidence metrics with predictions and maintaining human oversight in critical decision-making. Second, there are privacy implications in tracking specific aircraft, particularly for private and military flights. This could be mitigated through anonymization techniques and access controls for sensitive flight data. Finally, algorithmic bias may emerge if models are trained predominantly on data from certain regions or aircraft types, potentially leading to reduced performance for underrepresented categories. Regular audits of prediction accuracy across different aircraft categories and geographical regions could help identify and address such biases.

**Part B: Unsupervised Learning**

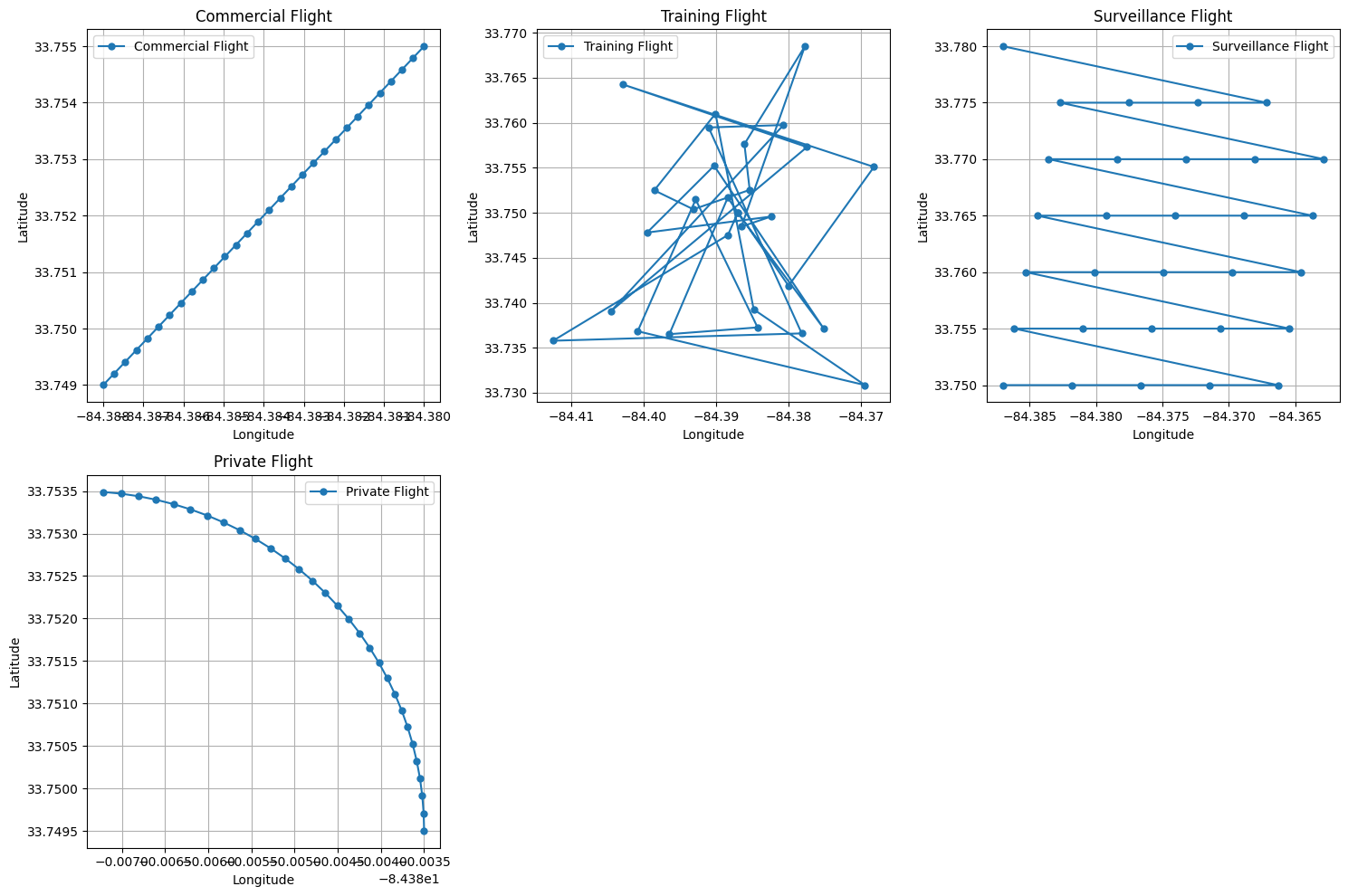
#### **Methods Description**

1. **Dynamic Time Warping (DTW)** measures similarity between two time-series data sequences by aligning them in a way that minimizes the distance between corresponding points. DTW can handle time-series data that may be out of phase or that vary in speed. In other words, it compares sequences that may have different lengths or sampling rates but still represent similar trends.
2. **K-means clustering** is an unsupervised machine learning algorithm that partitions a dataset into a specified number of clusters, where each data point belongs to the cluster whose centroid is nearest. The algorithm iteratively updates the centroids until convergence, aiming to minimize the within-cluster sum of squares (inertia).

#### **Method Comparisons**

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Prototype Matching (DTW)** | **K-Means Clustering** |
| **Methodology** | Direct comparison of flight path features to predefined prototypes. | Grouping flight paths based on similarity in feature space. |
| **Strengths** | Accurate for categorizing well-defined flight paths. | Can discover new patterns in flight paths without predefined labels. |
| **Weaknesses** | Sensitive to noise and small variations. | Assumes spherical clusters, may struggle with complex patterns. |
| **Suitability** | Best suited for well-known flight categories. | Best for discovering hidden patterns in data. |
| **Output** | Flight category labels based on closest prototype. | Grouped clusters representing different flight behaviors. |

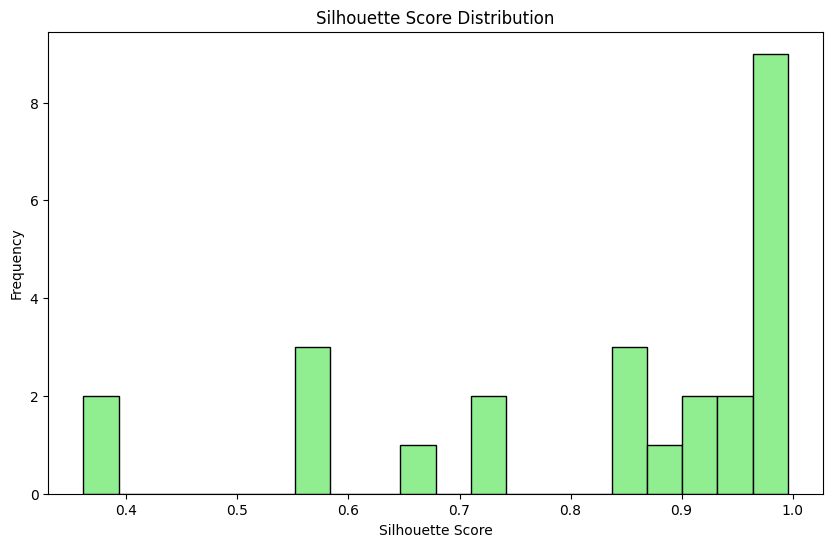
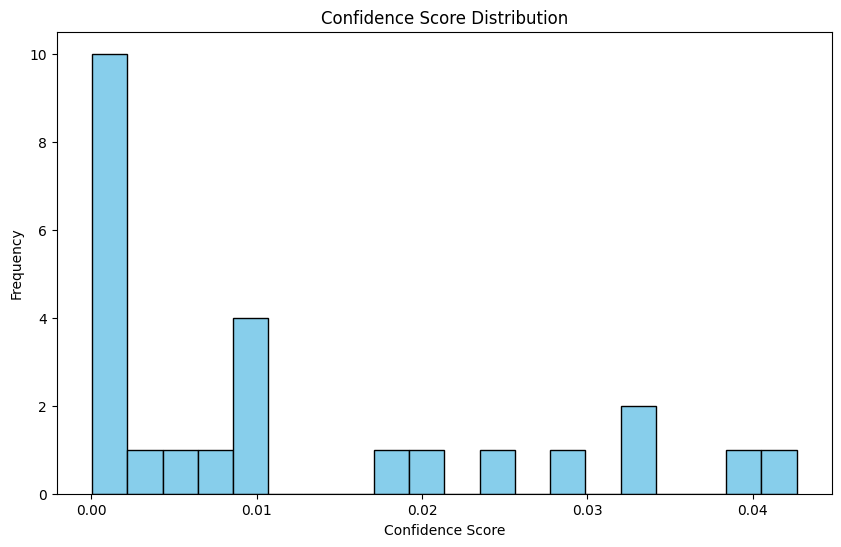
**Dynamic Time Warping**

1. **Prototype Selection**: Predefined prototype flight paths for each category (e.g., commercial flight, surveillance flight, training flight) are used as benchmarks. Prototypes:
2. **Flight Path Comparison**: For each observed flight path, the DTW distance between the flight path and each prototype is calculated. The prototype with the minimum DTW distance is selected as the best match.
3. **Clustering**: Based on the similarity measure, flights are grouped into clusters corresponding to the closest prototype. This is a typical unsupervised learning process where no prior labels are given, and the model tries to determine the structure in the data.

##### **Evaluation Metrics**

**Confidence score:** confident the model is in the classification of a given flight path. It reflects the degree of similarity between a flight path and its assigned prototype. A higher confidence score means that the model is more certain about the match, while a lower score indicates uncertainty.

**Silhouette score:** evaluates how well the clusters are separated. It measures the consistency of a flight path within its assigned cluster compared to other clusters. A high silhouette score means that the flight paths are well-separated into distinct groups, while a low silhouette score indicates that the flight paths may be poorly clustered.

**Confidence and Silhouette Statistics:**

|  |  |  |
| --- | --- | --- |
| Metric | Confidence Score | Silhouette Score |
| 25% | 0.000553 | 0.733166 |
| 50% | 0.006739 | 0.904332 |
| 75% | 0.019246 | 0.989859 |
| Max | 0.042665 | 0.995747 |
| Mean | 0.011962 | 0.826028 |
| Min | 0.000042 | 0.361291 |
| Std | 0.013783 | 0.200462 |

**Confidence Interpretation**: The confidence scores suggest that while the model is fairly confident in most of its classifications (with a mean of 0.011962), there is still significant variability, as evidenced by the standard deviation of 0.013783. This indicates that some flight paths have low confidence in their assigned labels.

**Silhouette Interpretation**: The silhouette score ranges from 0.361291 to 0.995747, with a mean of 0.826028. This shows that the flight types are generally well-separated, indicating that the clustering process has been successful in identifying distinct groups.

### **DTW Conclusions**

* **Cluster Separation**: The silhouette scores show that the flight paths are well-separated into distinct clusters, suggesting that the DTW algorithm is successful in identifying meaningful patterns in the data.
* **Confidence**: While the confidence scores vary, with a mean of 0.011962, there is some uncertainty in certain classifications, which could be addressed by refining the prototype data or adjusting DTW parameters.
* **Flight Type Distribution**: The most common flight type is "Surveillance," which represents 56% of the data, followed by "Private" and "Training" flight paths.

These results demonstrate the effectiveness of the unsupervised DTW-based classification method in grouping similar flight paths. The quality of the clustering, as indicated by the silhouette scores, suggests that the model can effectively distinguish between different types of flight paths, even with some uncertainty in confidence scores. Though the overall technique would benefit greatly from more prototypes to match with. For example, there could be several prototypes that would be useful to represent “Training” or “Emergency” flight profiles.

**Sensitivity Analysis**

DTW's performance is highly sensitive to prototype selection. Since it aligns new trajectories to these prototypes, poorly selected or insufficient prototypes can lead to inaccurate classifications, as evidenced by the variability in our confidence scores. Additionally, DTW is computationally expensive, particularly as the number of prototypes increases.

**Key Sensitivities:**

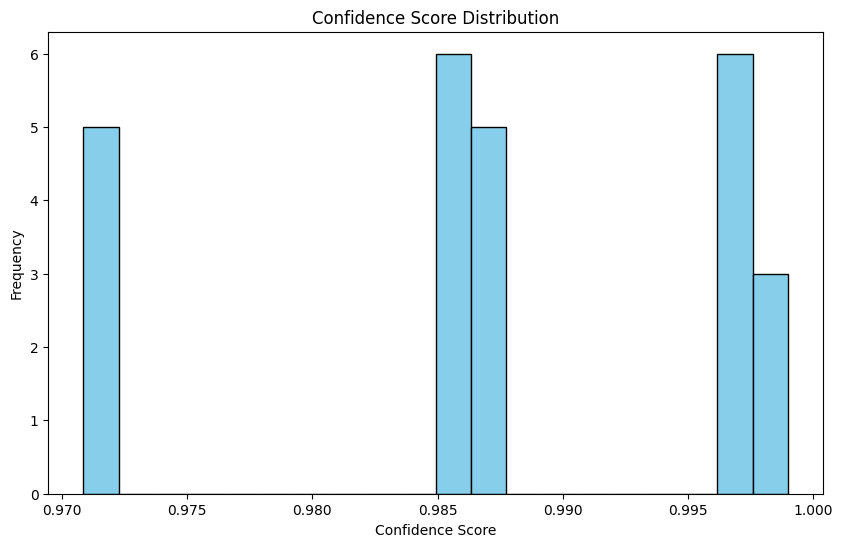
1. **Prototype Selection**: More or better-chosen prototypes improve classification but increase computational cost.
2. **Time Warping Window**: A larger warping window allows more flexibility in alignment but risks incorrectly matching dissimilar paths.
3. **Data Quality**: DTW struggles with noise in trajectory data, as misaligned or incomplete flight paths can distort similarity measures.

### **Next Steps**

* **Optimization**: Further tuning of the DTW parameters (such as the window size) and prototype selection could improve the confidence scores and reduce uncertainty.
* **Evaluation**: Additional techniques, such as manual validation or supervised learning methods, could be employed to verify and enhance the results of this unsupervised classification.

##### **2. K-Means Clustering**

### **Clustering Metrics**

* **Overall Silhouette Score:** 0.2935  
  This value indicates how well-separated the clusters are. A score closer to +1 means the clusters are well-separated, while a score near 0 indicates overlapping clusters. The silhouette score of 0.2935 suggests that the clustering results show some degree of separation but also some overlap or uncertainty in the classification.
* **Inertia:** 57.0456  
  Inertia measures the sum of squared distances between each data point and its assigned cluster's centroid. A lower inertia indicates better clustering. The inertia value of 57.0456 indicates the total "within-cluster" `distance across all points.

### **K-means Clustering Flight Classification Report**

### Statistical Summary of **Confidence Scores** and **Silhouette Scores**:

|  |  |  |
| --- | --- | --- |
| **Metric** | **Confidence Score** | **Silhouette Score** |
| **count** | 70188 | 70188 |
| **mean** | 0.9872 | 0.2893 |
| **std** | 0.0096 | 0.1601 |
| **min** | 0.9709 | -0.0062 |
| **25%** | 0.9853 | 0.1531 |
| **50%** | 0.9873 | 0.2644 |
| **75%** | 0.9975 | 0.3839 |
| **max** | 0.9990 | 0.5675 |

### **Interpretation of the K-means Clustering Results**

#### **Confidence Score:**

* The **mean** confidence score of 0.9872 suggests that, on average, the algorithm is highly confident in the classifications of the flight paths. A value close to 1 indicates strong confidence in the assignment of each flight to its respective cluster.
* The **standard deviation** of 0.0096 indicates a low degree of variability in the confidence scores across the dataset, which suggests that the model's confidence is generally consistent.
* The **minimum** confidence score of 0.9709 and **maximum** score of 0.9990 show that most flight paths have a high degree of confidence in their classification, though a few are classified with slightly less certainty.

#### **Silhouette Score:**

* The **mean** silhouette score of 0.2893 indicates that the clustering is somewhat moderate, with some overlap between clusters. A score closer to 0 suggests that the clusters are not perfectly well-separated, but the clustering is still somewhat meaningful.
* The **standard deviation** of 0.1601 highlights that there is a notable variation in the clustering quality, with some clusters being better-separated than others.
* The **minimum** silhouette score of -0.0062 indicates some data points might be poorly assigned to their clusters, possibly suggesting misclassifications or overlapping clusters.
* The **maximum** silhouette score of 0.5675 shows that some clusters are quite distinct and well-separated, indicating good clustering for those specific data points.

### **Conclusions**

1. **Cluster Separation:** The overall **Silhouette Score** of 0.2935 indicates that the flight paths are somewhat well-separated into distinct clusters, but there is still room for improvement. The moderate score suggests that some of the clusters might overlap or be ambiguous.
2. **Confidence in Classification:** The **Confidence Scores** are generally very high (mean of 0.9872), indicating that the K-means algorithm is confident in most of its classifications. However, a small number of flight paths may be less confidently classified, as indicated by the lower minimum confidence score (0.9709).
3. **Inertia:** The **Inertia** value of 57.0456 suggests that the clustering could be optimized further. Lowering inertia would imply that the data points are more tightly grouped within their clusters.
4. **Cluster Consistency:** The variation in the silhouette scores (standard deviation of 0.1601) indicates that there are clusters with varying degrees of separation. Some clusters are well-defined, while others might require further refinement.
5. **Drawbacks**: This technique suffers most in the mapping from clusters back to flight profiles. Categorizing a flight profile on simple conditional logic is challenging and could benefit greatly from a prototype matching step after the clustering step.

### **Next Steps**

1. **Optimization:** Tuning the number of clusters (K) or experimenting with different initialization methods for the K-means algorithm may improve the clustering results, particularly in terms of reducing inertia and increasing the silhouette score.
2. **Evaluation with Other Methods:** It may be helpful to compare the results of K-means with other clustering methods, such as DBSCAN or hierarchical clustering, to see if the clustering quality can be improved.
3. **Further Refinement:** Exploring feature engineering (e.g., adding additional attributes like time of day, weather conditions, etc.) could lead to better-defined clusters and higher confidence scores.

**Overall Conclusions on Unsupervised Learning:** The classification of flight data using unsupervised learning posed several challenges, primarily due to the lack of subject matter expertise and the absence of ground truth labels. Both approaches—Dynamic Time Warping (DTW)-based classification and K-means clustering—relied entirely on random data from the OpenSky network, limiting the ability to assess accuracy beyond statistical metrics such as F1 score and confusion matrices. While each method demonstrated unique strengths, their effectiveness was constrained by the inherent limitations of unsupervised learning. Incorporating a supervised step, such as a convolutional neural network (CNN), could have improved classification by providing more refined grouping and validation of flight profiles.

Between the two methods, DTW-based classification showed greater promise in terms of cluster separation, achieving a high silhouette score (0.826) by effectively aligning flight paths despite variations in speed and sampling intervals. However, its reliance on predefined prototypes introduced variability in confidence scores, highlighting the need for carefully curated prototypes—potentially developed with input from a subject matter expert. On the other hand, K-means clustering produced consistently high confidence scores (0.9872) but suffered from a lower silhouette score (0.2935), indicating overlapping clusters and potential misclassifications due to its inability to account for time-dependent variations. Visual inspection suggests that K-means performed better overall, but DTW could surpass it with an expanded set of well-chosen prototypes. Ultimately, while DTW demonstrated a stronger ability to capture nuanced flight path differences, refining both approaches—through improved prototype selection for DTW and possibly hybridizing with supervised learning—could lead to a more robust classification framework.

**Ethical Issues in Unsupervised Learning**

For flight path clustering, potential issues include misclassification of flight types leading to incorrect assumptions about aircraft operations. This risk can be reduced by communicating confidence scores with classifications. Clustering might also inadvertently reveal sensitive information about airport operations or airline routing strategies, requiring data aggregation protocols. Lastly, unsupervised approaches may create categories that reinforce existing assumptions rather than revealing novel patterns, which requires validation with domain experts.

**Statement of Work**

|  |  |  |
| --- | --- | --- |
| **Matthew** | **Enid** | **Julien** |
| Data Acquisition, Feature Definitions, Report Generation, Traffic python library utilization, Project Management | flight trajectory classification using unsupervised learning, specifically DTW-based classification and K-means clustering, on OpenSky Network data. Preprocessing flight paths, applying both methods, and evaluating their performance using silhouette and confidence scores. | Data Acquisition and Data Pipeline, creating representative flight samples. Supervised Learning - LSTM, Transformer, XGBoost. Evaluation, Sensitivity Analysis, Feature Analysis, Flight Forecasting Visualizations |
| All members of the team provided input to all portions of the report. The areas listed above were lead areas. | | |

# **Appendix**

## **A. Bibliography**

Vos, R. W., Sun, J., & Hoekstra, J. M. (2024). A Transformer-based Trajectory Prediction Model to Support Air Traffic Demand Forecasting. In E. Neiderman, M. Bourgois, D. Lovell, & H. Fricke (Eds.), Proceedings International Conference on Research in Air Transportation Article ICRAT 2024-87

Xingchen Dong, Yong Tian, Kexin Niu, Mengyuan Sun, Jiangchen Li, "Research on flight trajectory prediction method based on transformer," Proc. SPIE 13018, International Conference on Smart Transportation and City Engineering (STCE 2023), 1301854 (14 February 2024); https://doi.org/10.1117/12.3024772

## Song, Y., Yu, K., & Young, S. (2020). Categorizing Flight Paths using Data Visualization and Clustering Methodologies.RAT 2020. The Ohio State University. <https://arxiv.org/pdf/2310.00773>

## **B. Data Schema**

### **B.1 OpenSky Network ADSB Dataset**

For our supervised learning component, we primarily utilized the state\_vectors\_data4 table from the OpenSky Network ADSB dataset. The Automatic Dependent Surveillance-Broadcast (ADSB) is a surveillance technology in which aircraft broadcast their state information during flight, including position, altitude, speed, and identification data.

#### **OpenSky state\_vectors\_data4 Table Structure**

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Description** |
| time | integer | Unix timestamp when the state vector was recorded |
| icao24 | varchar | Unique ICAO 24-bit address of the aircraft |
| lat | double | Latitude of the aircraft in decimal degrees |
| lon | double | Longitude of the aircraft in decimal degrees |
| velocity | double | Ground speed of the aircraft in meters per second |
| heading | double | Heading of the aircraft in decimal degrees (0-360) |
| vertrate | double | Vertical rate in meters per second (positive for climbing) |
| callsign | varchar | Callsign of the aircraft |
| onground | boolean | Flag indicating if the aircraft is on the ground |
| alert | boolean | Flag indicating if the aircraft has alert status |
| spi | boolean | Special Position Identification flag |
| squawk | varchar | Transponder code (octal) |
| baroaltitude | double | Barometric altitude in meters |
| geoaltitude | double | Geometric altitude in meters |
| lastposupdate | double | Unix timestamp when position was last updated |
| lastcontact | double | Unix timestamp when the aircraft was last seen |
| serials | array(integer) | Array of receiver IDs that received this message |
| hour | integer | Hour partition key for database optimization |

This table is partitioned by hour to optimize query performance for time-based operations. Each record represents a complete state vector of an aircraft at a specific point in time, providing all necessary parameters for trajectory prediction.

### **B.2 Complete OpenSky Database Structure**

The OpenSky Network database contains several related tables capturing different aspects of aviation data:

* **flarm\_raw**: This table stores raw FLARM/OGN sensor data, capturing details like sensor location (latitude, longitude, altitude), timestamps (server, sensor, and plane times), raw message content (26-byte FLARMv6 messages), and signal metrics (frequency, SNR, CRC validation). It includes technical corrections (e.g., NTP clock error, frequency adjustments) and distinguishes between OGN/FLARM message types. Data is partitioned by hour for efficient querying.
* **flights\_data4**: Tracks flight trajectories and airport estimates, including ICAO24 identifiers, first/last seen timestamps, departure/arrival airport candidates, and a track array storing positional waypoints (time, latitude, longitude). Metrics like horizontal/vertical distances to estimated airports and candidate airport lists are included. Partitioned by day.
* **flights\_data5**: An extended version of flights\_data4, adding precise takeoff/landing details (times, coordinates), confirmed departure/destination airports, and expanded flight phase metadata. Like its predecessor, it uses day partitioning and retains core flight tracking features.
* **identification\_data4**: Focuses on aircraft identification, linking ICAO24 codes to emitter categories, flight identity codes, and message metadata. Includes sensor arrays contributing to detection and raw message content. Partitioned by hour.
* **operational\_status\_data4**: Captures detailed aircraft system statuses, including TCAS capabilities, GPS accuracy metrics (NAC, NIC), antenna configurations, and safety flags (e.g., low TX power, TCAS advisories). Technical fields like systemdesignassurance and airplanelength/width provide operational context. Partitioned by hour.
* **position\_data4**: Stores real-time positional data (latitude, longitude, altitude) with quality indicators like NIC/NAC codes and surveillance status. Includes groundspeed, heading, and surface detection flags. Sensor arrays track contributing receivers. Partitioned by hour.
* **rollcall\_replies\_data4**: Records aircraft responses to interrogations (e.g., ATC radar), including flight status, altitude, identity codes, and utility messages. Fields like interrogatorid and downlinkrequest clarify the context of replies. Partitioned by hour.
* **velocity\_data4**: Details velocity components (NS/EW speed, vertical rate) and motion attributes like supersonic flags, intent changes, and heading. Includes navigation accuracy (NAC) and barometric/geometric altitude references. Partitioned by hour.

Each table is partitioned chronologically (hour or day) to optimize performance, with sensor/aircraft metadata, raw messages, and derived metrics tailored to specific aviation data use cases (e.g., tracking, identification, operational monitoring).

**B.3 Data Preprocessing Pipeline**

For our predictive models, we implemented a sophisticated data preprocessing pipeline leveraging the OpenSky Network ADSB flight data. The following stages were implemented:

* **Data Acquisition**: We developed specialized query modules (FlightQueries and StateVectorQueries) to efficiently retrieve flight data from the OpenSky database, with support for both time-based chunking and random sampling strategies as implemented in interval\_generation.py.
* **Geographical Filtering**: State vector queries were constrained to specific geographic bounds (primarily focused on Georgia airspace) to analyze regional flight patterns, as defined in the GEORGIA\_BOUNDS constant.
* **Track Metrics Computation**: For flight trajectories in the track column, we computed comprehensive aggregate metrics including min/max/mean values for latitude, longitude, altitude, heading, and time parameters using the compute\_track\_metrics function.
* **Time-based Resampling**: To address the irregular sampling intervals in the original data, we implemented a consistent time-based resampling strategy using resample\_flight\_state\_data, which converts timestamps to datetime objects and resamples at configurable intervals (default 5-second intervals).
* **Sequential Processing**: Data was organized into chronological order by timestamp before any model training, ensuring temporal continuity in trajectory analysis.
* **Preprocessing Pipeline Optimization**: The pipeline was designed with efficiency in mind, implementing features like conditional processing with skip\_if\_exists to avoid redundant computations and file operations.
* **Data Persistence Strategy**: Processed data was systematically stored in Parquet format to optimize storage space and retrieval performance, with appropriate error handling and logging throughout the process.

This comprehensive preprocessing approach ensured that our models received high-quality, consistently formatted flight trajectory data, which was critical for the accurate prediction of aircraft movement patterns and states.

**B.4 Feature Engineering**

**B.4.1 Transformer/LSTM Base Features:** For our sequence-based models (Transformer, LSTM, FFNN), we utilized a core set of six raw features from the aircraft state data without complex feature engineering:

|  |  |
| --- | --- |
| **Feature** | **Description** |
| lon | Longitude of the aircraft in decimal degrees |
| lat | Latitude of the aircraft in decimal degrees |
| heading | Heading of the aircraft in decimal degrees (0-360) |
| velocity | Ground speed of the aircraft in meters per second |
| vertrate | Vertical rate in meters per second (positive for climbing) |
| geoaltitude | Geometric altitude in meters |

These features were organized into temporal sequences with a fixed window size (input\_sequence\_length=44 by default), preserving the inherent temporal relationships in the data. This approach leverages the ability of Transformer and LSTM architectures to automatically learn temporal patterns and dependencies across the sequence.

**B.4.2 XGBoost Engineered Features:** For the XGBoost model, which does not inherently handle sequential data, we implemented extensive feature engineering to capture temporal patterns. For each of the six base features, we derived multiple engineered features:

1. **Last Values**:
   * {feature}\_last - The most recent value for each base feature
2. **Statistical Features**:
   * {feature}\_mean - Mean value over the sequence
   * {feature}\_std - Standard deviation over the sequence
   * {feature}\_min - Minimum value in the sequence
   * {feature}\_max - Maximum value in the sequence
3. **Rate of Change Features**:
   * {feature}\_delta1 - Change between last two timepoints
   * {feature}\_delta2 - Change over last three timepoints
   * {feature}\_delta5 - Change over last six timepoints (when sequence length permits)
4. **Acceleration Features**:
   * {feature}\_accel - Second derivative (change in rate of change)

This feature engineering process transformed the original 6-feature sequential data into a rich, 24+ feature tabular dataset that enabled XGBoost to effectively capture temporal patterns despite not being explicitly designed for sequence modeling. The engineered features provide explicit representations of statistical properties, trends, and dynamics that sequence models must learn implicitly.

## **C. Code Structure**

Complete project code is available at: https://github.com/JewelsHovan/Aviation-Milestone2-Project

Sample data for each OpenSky database table described in section B can be found in the 'samples' directory of the GitHub repository.

### **C.1 Data Pipelines**

These modules handle SQL queries and data retrieval from OpenSky:

* **Flight Pipeline**: src/pipeline/flight\_pipeline.py  
  + Handles retrieving flight data in chunks or by sampling
  + Includes FlightsPipeline class with methods for different retrieval strategies
* **State Vector Pipeline**: src/pipeline/state\_vector\_pipeline.py  
  + Similar structure to flight pipeline but focused on state vector data
  + Contains StateVectorPipeline class with chunking and sampling methods

### **C.2 Query Modules**

SQL query generation for OpenSky database:

* **Flight Queries**: src/queries/flight\_queries.py  
  + Contains FlightQueries class with methods to generate SQL for flights data
  + Includes queries for flight data versions 4 and 5, as well as ICAO-specific queries
* **State Vector Queries**: src/queries/state\_vector\_queries.py  
  + Contains StateVectorQueries class with methods for state vector retrieval
  + Provides geographic bounding and ICAO-specific queries

### **C.3 Retrieval Utilities**

* **Interval Generation**: src/retrieval/interval\_generation.py  
  + Functions to generate time intervals for data retrieval
  + Includes methods for both chunk-based and random sampling approaches
* **Retrieval Engine**: src/retrieval/retrieval\_engine.py  
  + Core function to retrieve data using provided intervals and query functions
  + Handles the actual execution of queries against OpenSky Trino database

### **C.4 Transformation Modules**

* **Flight Preprocessing**: src/transformations/flight\_preprocessing.py  
  + Functions to compute metrics from flight tracks and preprocess flight data
  + Provides functions to extract statistical features from raw trajectory data
* **State Preprocessing**: src/transformations/state\_preprocessing.py  
  + Contains functions to resample flight state data at specified intervals
  + Ensures consistent sampling rates for model input

### **C.5 Model Architecture**

* **Model Definitions**: state\_prediction/models.py Contains all model architectures:
  + TransformerPredictor: Transformer-based sequence predictor
  + LSTMPredictor: LSTM-based sequence predictor
  + FFNNPredictor: Feed-forward neural network predictor
  + KalmanFilterPredictor: Kalman filter-based predictor
  + Factory function get\_model() to instantiate the appropriate model

### **C.6 Training Scripts**

* **Generic Model Training**: state\_prediction/scripts/train.py  
  + Main training script that handles neural network-based models
  + Includes full training loop with validation, early stopping, and checkpointing
* **XGBoost Training**: state\_prediction/scripts/train\_xgboost.py  
  + Specialized script for training XGBoost models
  + Includes feature engineering specific to XGBoost approach

### **C.7 Prediction and Evaluation**

* **Prediction Script**: state\_prediction/scripts/predict.py  
  + Main script for generating predictions from trained models
  + Can perform single-step prediction or multi-step autoregressive generation
* **XGBoost Prediction**: state\_prediction/scripts/predict\_xgboost.py  
  + Specialized prediction script for XGBoost models
  + Handles the feature engineering required for XGBoost
* **Evaluation Script**: state\_prediction/scripts/evaluate.py  
  + Evaluates models using k-fold cross-validation
  + Calculates multiple error metrics (MSE, RMSE, MAE)
* **XGBoost Evaluation**: state\_prediction/scripts/evaluate\_xgboost.py  
  + Specialized evaluation script for XGBoost models
  + Allows evaluation of individual target variables or all targets

### **C.8 Analysis Scripts**

* **Model Failure Analysis**: state\_prediction/scripts/analyze\_model\_failures.py  
  + Analyzes cases where the model predictions have high error
  + Uses SHAP values to understand feature contributions to errors
* **XGBoost Failure Analysis**: state\_prediction/scripts/analyze\_xgboost\_failures.py  
  + Similar to the above but specialized for XGBoost models
* **Feature Analysis**: state\_prediction/scripts/feature\_analysis.py  
  + Performs feature importance analysis and ablation studies
  + Helps understand which features contribute most to model performance
* **Sensitivity Analysis**: state\_prediction/scripts/sensitivity\_analysis.py  
  + Analyzes how model performance varies with hyperparameter changes
  + Particularly focused on LSTM hyperparameters
* **Tradeoff Analysis**: state\_prediction/scripts/tradeoff\_analysis.py  
  + Analyzes tradeoffs between different model characteristics
  + Includes data size vs. accuracy, speed vs. accuracy, and model capacity tradeoffs

### **C.9 Data Preparation and Visualization**

* **Data Preparation**: state\_prediction/scripts/prepare\_data.py  
  + Handles preprocessing of raw flight data for model training
  + Creates sliding windows, splits into train/test, and scales the data
* **Flight Visualization**: state\_prediction/scripts/visualize\_flights.py  
  + Creates visualizations of flight trajectories and predictions
  + Includes both simple and enhanced visualization styles
* **Single Flight Forecast**: state\_prediction/scripts/forecast\_single\_flight.py  
  + Predicts and visualizes trajectory for a single flight
  + Useful for case-study analysis of model performance

### **C.10 Configuration and Utilities**

* **Configuration**: state\_prediction/scripts/config.py  
  + Contains all configuration parameters for the project
  + Includes data, model, training, and path configurations
* **Path Utilities**: src/utils/paths.py and state\_prediction/scripts/paths.py  
  + Define project directory structure
  + Ensure consistent file paths across the project
* **Constants**: src/utils/constants.py  
  + Defines constants like geographic bounds and airport information
* **File Utilities**: src/utils/file\_utils.py  
  + Functions for file operations, particularly saving to CSV and Parquet formats
* **Time Utilities**: src/utils/time\_utils.py  
  + Functions for parsing dates and sampling random datetimes

## **D. Key Visualizations**

### **D.1 Model Performance Visualizations**

* Training loss curves: Located in each model's directory under state\_prediction/model/<model\_type>/visualizations/
* Cross-validation metrics: state\_prediction/model/<model\_type>/visualizations/<model\_type>\_cv\_metrics.png

### **D.2 Feature Importance Visualizations**

* SHAP summary plots: state\_prediction/model/xgboost/<target>\_shap\_summary.png
* Feature importance bar charts: state\_prediction/model/xgboost/<target>\_feature\_importance.png
* Ablation analysis results: state\_prediction/model/<model\_type>/visualizations/<model\_type>\_feature\_importance.png

### **D.3 Flight Trajectory Visualizations**

* Multiple flight paths: state\_prediction/model/visualizations/multiple\_flight\_paths.png
* Enhanced flight visualization: state\_prediction/model/visualizations/enhanced\_flight\_paths.png
* Single flight forecast: state\_prediction/model/<model\_type>/visualizations/single\_flight\_forecast\_<model\_type>.png

### **D.4 Sensitivity Analysis Visualizations**

* Parameter sensitivity: state\_prediction/model/lstm/visualizations/sensitivity/lstm\_<param>\_sensitivity.png
* Learning curves: state\_prediction/model/lstm/visualizations/sensitivity/lstm\_<param>\_learning\_curves.png
* Comprehensive summary: state\_prediction/model/lstm/visualizations/sensitivity/lstm\_sensitivity\_summary.png

### **D.5 Tradeoff Analysis Visualizations**

* Data size tradeoffs: state\_prediction/model/<model\_type>/visualizations/tradeoffs/<model\_type>\_data\_size\_tradeoff.png
* Speed-accuracy tradeoffs: state\_prediction/model/<model\_type>/visualizations/tradeoffs/<model\_type>\_speed\_accuracy\_tradeoff.png
* Model capacity tradeoffs: state\_prediction/model/<model\_type>/visualizations/tradeoffs/<model\_type>\_model\_capacity\_tradeoff.png

## **E. Model Results**

### **E.1 Evaluation Results**

* Model evaluation summaries: state\_prediction/model/<model\_type>\_evaluation\_results.json
* XGBoost evaluation results: state\_prediction/model/xgboost/<target>\_evaluation\_results.json

### **E.2 Model Analysis**

* Failure analysis results: state\_prediction/model/<model\_type>/failure\_analysis/
* XGBoost failure analysis: state\_prediction/model/xgboost/failure\_analysis/<target>/

### **E.3 Trained Models**

* Neural network models: state\_prediction/model/<model\_type>/<model\_type>\_best\_model.pth
* XGBoost models: state\_prediction/model/xgboost/<target>\_model.bin
* Model configurations: state\_prediction/model/<model\_type>/<model\_type>\_config.json

## **F. Data Pipeline Workflow**

1. **Data Retrieval**:  
   * Use FlightsPipeline or StateVectorPipeline to retrieve data from OpenSky
   * Data is stored in Parquet format in the specified output directory
2. **Data Preprocessing**:  
   * prepare\_data.py handles resampling, windowing, and scaling
   * Processed data and scalers are saved in state\_prediction/model/train\_data/ and state\_prediction/model/scalers/
3. **Model Training**:  
   * Use train.py or train\_xgboost.py to train models on preprocessed data
   * Models are saved in their respective directories
4. **Model Evaluation and Analysis**:  
   * Various evaluation and analysis scripts can be run on trained models
   * Results and visualizations are saved in appropriate directories