Big Data Insights into U.S. Flight Anomalies: From Large-Scale Batch Analytics to Real-Time Delay Prediction

Aakash Vardhan Madabhushi, Aishwarya Iyer, Keith Rajesh Gonsalves, Om Dankhara
Department of Applied Data Science, San Jose State University
GitHub Repository

***Abstract*—Commercial aviation in the United States operates at a scale where even modest inefficiencies accumulate into substantial operational and economic consequences. Flight delays, in particular, remain a persistent challenge for airlines, airports, and passengers, influencing crew scheduling, gate assignments, network flow, and customer satisfaction. At the same time, modern aviation systems generate vast volumes of structured and semi-structured data, much of which is underutilized for proactive, data-driven decision support. This project explores how contemporary Big Data ecosystems can transform large, noisy, real-world datasets into actionable insights and predictive tools that assist both historical analysis and real-time operations.**

We analyze a 15-month portion of the U.S. Bureau of Transportation Statistics (BTS) "Reporting Carrier On-Time Performance" dataset, consisting of approximately 9.5 million flight records distributed across fifteen monthly CSV files. Using Apache Spark, we construct a scalable batch-processing workflow that performs schema normalization, timestamp correction, missing-value handling, outlier mitigation, and extensive feature engineering tailored to temporal, operational, and route-level characteristics. These processing steps ensure that the multi-gigabyte dataset can be queried, transformed, and modeled efficiently on distributed infrastructure.

Exploratory Data Analysis (EDA) uncovers clear weekly and seasonal patterns in delays, variations across airlines and airports, and meaningful relationships between delay causes such as weather, late aircraft, and NAS-related disruptions. Guided by these insights, we develop and evaluate three supervised machine learning models—logistic regression, decision tree, and random forest—within Spark MLlib. Performance on a stratified test split indicates that the random forest model is the most effective, achieving an accuracy near 79% and an F_1-score of approximately 0.71 while demonstrating limited overfitting and interpretable feature importance rankings.

To bridge historical learning with live operational contexts, we deploy the trained model within a Kafka–Spark Structured Streaming pipeline capable of processing live-like flight events and outputting real-time delay probabilities to a monitoring dashboard. The project illustrates the end-to-end integration of Big Data tools for aviation analytics, highlights practical engineering considerations, and demonstrates how large-scale batch pipelines can evolve into streaming systems that support timely decision-making.

***Index Terms*—Flight delays, Big Data, Apache Spark, Apache Kafka, Spark MLlib, streaming analytics, transportation data.**

I. Introduction

Flight delays are a familiar nuisance to travelers and a persistent headache for airlines and airports. Beyond individual inconvenience, delays propagate through tightly coupled schedules, increasing crew costs, disrupting aircraft rotations, and contributing to airport congestion. Industry and academic studies estimate that the annual cost of flight delays in the United States reaches several billions of dollars when passenger time, operating expenses, and missed connections are counted together [1].

At the same time, airlines and regulators now collect increasingly detailed operational data. The BTS "Reporting Carrier On-Time Performance" dataset provides a longitudinal record of every scheduled flight by participating carriers, including timestamps, route information, delay indicators, and breakdowns of delay minutes into categories such as carrier, weather, and late aircraft. The dataset is publicly available, but its size and complexity make naive, single-machine analysis slow and fragile.

This project asks a practical question: *Can Big Data tools commonly taught in a graduate data engineering course be used to build a scalable system that both analyzes historical patterns and serves real-time delay predictions?* Rather than treating batch analytics and streaming as separate worlds, we experiment with a single architecture that moves from historical learning to online inference.

We focus on a 15-month slice of BTS data from April 2024 through June 2025. After cleaning and integrating the monthly files, we treat delay prediction as a binary classification problem: predict whether a flight will arrive at least fifteen minutes late. We then design a pipeline that uses Apache Spark for batch processing and Apache Kafka with Spark Structured Streaming for scoring incoming flight events. The report is intentionally descriptive; each step, figure, and design choice is explained thoroughly so that another practitioner could reproduce or adapt the system.

II. Dataset and Big Data Characteristics

A. Source and Structure

The raw data come from the BTS "Reporting Carrier On-Time Performance" archive [2]. Each month is supplied as a CSV file with one row per scheduled flight and roughly forty-seven columns. For this project we obtained fifteen files spanning April 2024 to June 2025. Figure 1 (a cropped screenshot from our working directory) lists a subset of those files with approximate sizes; most lie between 140 MB and 165 MB in compressed form.

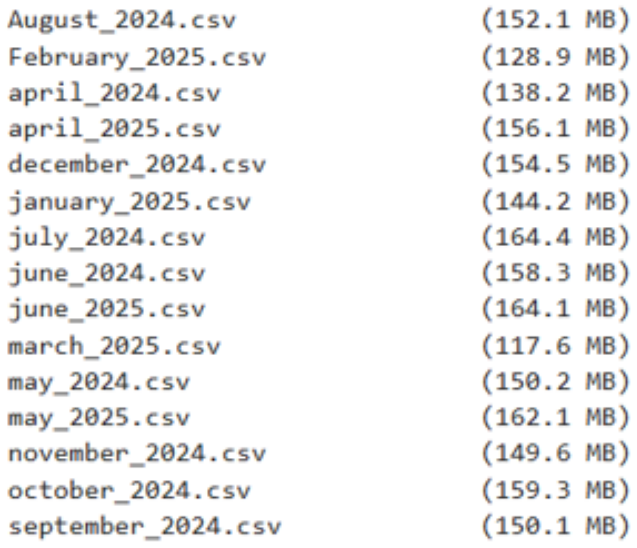

Fig. 1: Excerpt of BTS monthly CSV files used in the project, with sizes ranging from roughly 117 MB to 165 MB per file.

When loaded into Spark with appropriate schemas, the fifteen files together yield around 9.5 million records. Each record includes flight date, carrier, origin and destination airport codes, scheduled and actual departure and arrival times, distance, taxi times, various indicator variables (e.g., whether a flight was cancelled or diverted), and several delay-minute columns indicating how much of the delay is attributed to carrier, weather, National Airspace System (NAS), security, or late aircraft causes.

B. Big Data "V" Dimensions

The dataset exhibits several classic Big Data characteristics:

Volume. The raw files together occupy around 3.1 GB on disk. While this is small by industrial standards, it is large enough that repeated full scans and joins become slow on a single laptop, especially when performing group-by aggregations for EDA. Spark's ability to distribute the work across partitions significantly reduces runtime.

Velocity. The BTS archive is historical, but the schema mirrors fields that airlines generate in near real time. Part of our goal is to show how the same structure can be used in a streaming setting by replaying flights through Kafka at a controlled rate.

Variety. The data include categorical identifiers (carriers, airport codes), numeric durations, dates and times, indicator flags, and several derived delay metrics. Types are not always consistent across months, and certain columns (such as cancellation codes) are sparsely populated. This variety leads to nontrivial cleaning and encoding choices.

Taken together, these properties justify the use of distributed tools. Traditional spreadsheet-style exploration is quickly overwhelmed by the number of rows, and even plain Python with pandas struggles with repeated passes over the full dataset.

C. Problem Definition

BTS provides two related delay indicators: `ARR_DEL15` and `DEP_DEL15`, signalling whether a flight arrived or departed fifteen or more minutes late. We frame our prediction problem around `ARR_DEL15` because arrival performance is more visible to passengers and more directly tied to missed connections.

Formally, for each flight i, we observe a feature vector $\mathbf{x}_i$ containing carrier, origin, destination, scheduled departure time, and other pre-departure attributes. The binary label y_i is 1 if the arrival delay is at least 15 minutes and 0 otherwise. Given historical data $\{(\mathbf{x}_i, y_i)\}$, our aim is to learn a function $f(\mathbf{x})$ that estimates $\Pr(y = 1 \mid \mathbf{x})$. The probabilistic output can be thresholded for classification or used directly for ranking and risk monitoring.

A practical constraint is that only information known at or before scheduled departure should be used. For example, actual departure time or intermediate delay minutes would not be available when making an early prediction. Throughout our pipeline we therefore restrict the feature set accordingly.

III. Data Processing and Feature Engineering

Processing 9.5 million rows is straightforward for Spark but still requires care to avoid hidden pitfalls such as schema drift, inconsistent time formats, and skewed partitions.

A. Schema Normalization and Ingestion

We begin by defining an explicit Spark schema for the most important columns. Using a fixed schema avoids relying on per-file type inference, which can be brittle when a column happens to be all nulls in a particular month. For example, `CRS_DEP_TIME` and `CRS_ARR_TIME` are stored as integers of the form `HHMM`. We cast them into strings and then into timestamps by combining them with the flight date, yielding a consistent representation across months.

During ingestion we also apply sanity checks. Flights with missing carrier codes, missing origin or destination airports, or obviously nonsensical times (e.g., a negative taxi-out duration) are flagged and either repaired or removed after manual inspection. Although these records are a small fraction of the total, cleaning them early prevents obscure downstream errors.

B. Handling Missing Values and Outliers

Delay-minute fields often contain zeros, but a zero can mean either "no delay in this category" or "delay minutes were not recorded." We treat values as missing when both the indicator flag and the minute column suggest inconsistency. For missing values where a clear interpretation exists—for instance, a flight not classified as weather-delayed but with a null weather delay minute—we impute zeros. For truly unknown fields we keep an explicit missing indicator so that models can learn whether the presence of missingness itself carries information.

Outliers in delay minutes and in arrival/departure times are handled conservatively. Extremely long delays (several thousand minutes) are rare and may reflect data entry issues. We cap delay minutes at a high quantile (e.g., the 99.5th

percentile) to limit their influence while preserving order. Similarly, flights with scheduled or actual times outside plausible ranges are removed.

C. Feature Engineering

The feature engineering stage transforms raw BTS columns into variables that are both predictive and computationally manageable in Spark MLlib.

1) Temporal Features: We extract month, day of week (1=Monday,. . . ,7=Sunday), hour of scheduled departure, and hour of scheduled arrival. For interpretability and to capture nonlinear patterns, we also group hours into coarse parts of day: early morning (00:00–05:59), morning bank (06:00–10:59), midday (11:00–15:59), evening (16:00–20:59), and late night (21:00–23:59). Additional flags mark holidays and days near the start or end of the month.

2) Route and Airport Features: Origin and destination airport codes are used directly as high-cardinality categoricals. We also create a route identifier by concatenating origin and destination. For some EDA tasks we compute historical on-time performance per route; these statistics are not directly used as features to avoid label leakage, but they help contextualize the model results.

3) Operational and Contextual Features: Distance is bucketed into short-, medium-, and long-haul groups to reflect different flight profiles. We also include indicators for flight type (e.g., redeye, cross-country) based on combinations of distance and scheduled time. Where available, we derive simple congestion proxies such as the number of other flights scheduled from the same origin within a fifteen-minute window.

All categorical variables are encoded using Spark's `StringIndexer` followed by `OneHotEncoder`, and then joined with numeric features using a `VectorAssembler`. This pipeline ensures that the same transformations can be applied consistently during training and streaming inference.

IV. Exploratory Data Analysis

EDA serves two roles in the project. First, it confirms that the data cleaning and feature engineering steps behave as intended. Second, it surfaces patterns that influence how we structure the machine learning models. All visualizations in this section are derived from Spark aggregations exported to Python plotting libraries.

A. Average Arrival Delay by Day of Week

Figure 2 plots the average arrival delay (in minutes) by day of week, where 1 denotes Monday and 7 denotes Sunday. Each bar summarises millions of flights across the 15-month period.

Two observations stand out. First, Friday (day 5) and Sunday (day 7) have visibly higher average delays than midweek days. This pattern aligns with intuition: Fridays carry heavy outbound leisure and business travel, while Sundays capture the return wave. Both days experience tight turnarounds and crowded schedules, so even small perturbations propagate

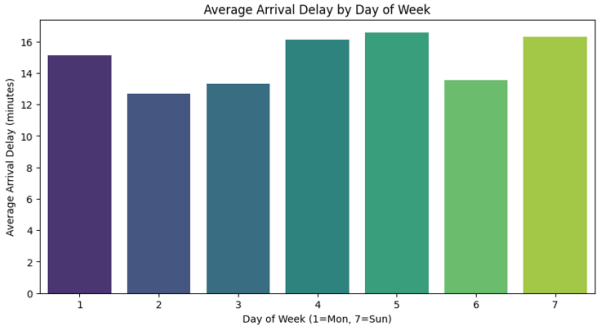

Fig. 2: Average arrival delay by day of week (1 = Monday, 7 = Sunday).

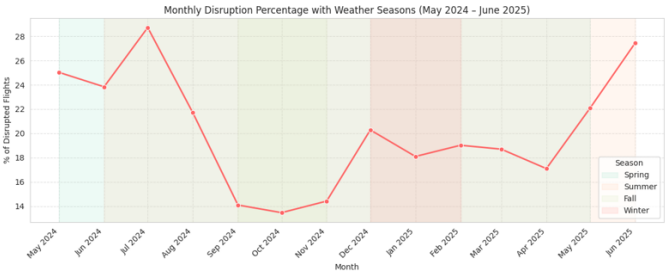

Fig. 3: Monthly percentage of disrupted flights with seasonal shading (May 2024–June 2025).

quickly. Second, Tuesday and Wednesday tend to have relatively lower delays, suggesting that flights in the middle of the week operate under somewhat more relaxed conditions.

From a modeling perspective, Figure 2 justifies including both raw day-of-week indicators and a coarser weekend flag. It also suggests that interactions between day of week and specific airports could be important, especially for hubs that serve a high concentration of leisure routes.

B. Seasonal Disruption Patterns

Delays vary not only by weekday but also by season. To quantify this effect we compute, for each month, the proportion of flights that were "disrupted" in the sense of being significantly delayed, cancelled, or diverted. Figure 3 displays this monthly disruption percentage from May 2024 through June 2025. The background shading indicates meteorological seasons: spring, summer, fall, and winter.

The curve reveals a pronounced spike in July 2024, where nearly 29% of flights in the sample are disrupted. This peak coincides with the height of the summer travel season and convective weather in many U.S. regions. Another notable increase appears in late winter; disruption rates rise again around December and remain elevated into early spring, reflecting snowstorms and holiday congestion. In contrast, September and October show comparatively calmer operations.

These seasonal patterns reinforce the value of month and season features in the model. They also highlight that any evaluation based on a fixed calendar window may over- or under-estimate performance depending on which months are included. In production, retraining schedules would need to account for such seasonality.

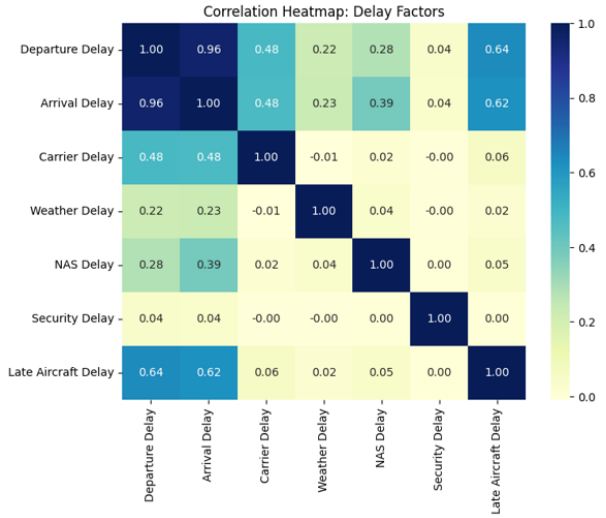

Fig. 4: Correlation heatmap among total departure/arrival delay and specific delay causes.

C. Relationships Among Delay Causes

The BTS dataset breaks down delay minutes into several categories. To understand how these components interact, we compute pairwise correlations among the main delay variables: departure delay, arrival delay, carrier delay, weather delay, NAS delay, security delay, and late aircraft delay. Figure 4 shows a correlation heatmap with values displayed in each cell.

Several patterns emerge. Departure and arrival delay are, unsurprisingly, almost perfectly correlated; once a flight pushes back late, it rarely recovers all the lost time, and downstream airport congestion often exacerbates the delay. Both are moderately correlated with late aircraft delay minutes, confirming that inbound lateness is a major driver of outbound performance. Weather and NAS delays show weaker correlations. Weather-related disruptions are less frequent but tend to be severe when they occur, while NAS delays reflect broader airspace constraints that can affect many flights simultaneously. Security delay is almost uncorrelated with the other causes and plays a minor role in aggregate.

The heatmap suggests that while cause categories are informative, they are also somewhat redundant with overall departure and arrival delay minutes. To avoid multicollinearity in the models, we use them as auxiliary EDA variables rather than direct predictors of `ARR_DEL15`. Instead, we rely more heavily on upstream features such as route, time of day, and historical congestion.

D. Airline-Level Differences

Passengers often perceive certain airlines as "always late" or "usually on time." To examine whether such perceptions have empirical support, we compute the average arrival delay for each carrier and then rank the top ten worst-performing airlines in terms of mean delay. Figure 5 displays the resulting bar chart.

A few carriers, particularly regional operators and low-cost airlines with dense schedules, show average delays exceeding

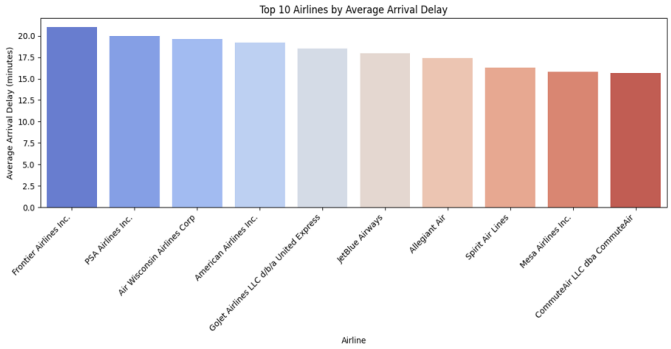

Fig. 5: Top ten airlines by average arrival delay across the 15-month window.

twenty minutes. Others cluster closer to the overall mean. It is important not to over-interpret these differences as pure airline "quality." Carriers with more complex hub-and-spoke networks or more exposure to weather-sensitive regions naturally face tougher operating conditions. However, Figure 5 confirms that carrier identity does carry predictive signal and should be included as a feature.

Overall, the EDA phase confirms that the engineered variables—day of week, month, season, carrier, origin, destination, and distance band—capture meaningful structure in the data. It also hints at complex interactions that motivate the use of nonlinear models such as decision trees and random forests.

V. Big Data Tools and System Architecture

A. Overview of Technologies

The project is intentionally built around tools that are widely used in industry and appear across Big Data curricula:

- **Apache Spark** for distributed batch processing, EDA, and machine learning.
- **Spark SQL** for relational-style aggregations and joins at scale.
- **Spark MLlib** for constructing end-to-end machine learning pipelines.
- **Apache Kafka** for durable, high-throughput message streaming.
- **Spark Structured Streaming** for consuming Kafka topics and scoring records in near real time.

Each tool contributes a specific capability, but their real value lies in how they integrate. A single Spark session can access batch data in cloud storage, train models, and then be repurposed as a streaming job reading from Kafka.

B. End-to-End Pipeline

Figure 6 presents the high-level pipeline diagram used in our presentation. Although visually simple, it encodes the flow from raw data to streaming predictions.

Starting at the top, the "Raw BTS Dataset" box represents the monthly CSV files. These are ingested into Spark, where cleaning and feature engineering produce a curated table suitable for both EDA and modeling. The next layer, "EDA + Visual Analysis," indicates our use of Spark SQL and Python-based plotting to examine distributional patterns. The

Fig. 6: End-to-end Big Data pipeline. Raw BTS data undergo cleaning and feature engineering in Spark, feed EDA and model training, and ultimately support a Kafka–Spark streaming loop that produces real-time delay predictions.

"ML Training & Validation" stage corresponds to the MLlib pipelines described in the next section. We serialize the best-performing model and register it as a reusable artifact. In the streaming half of the diagram, a Kafka producer sends live-like flight events to a topic. Spark Structured Streaming subscribes to that topic, reconstructs the same feature vector used during training, and applies the serialized model to generate predictions. Finally, predictions are written to downstream sinks and dashboards.

C. Spark and Kafka in Practice

Figure 7 is a screenshot from our development environment showing two terminal panes. On the left, a Python script acts as a Kafka producer, repeatedly reading batches of flights from the May 2025 file and publishing them to a topic at 100-record intervals. Log messages indicate the batch number and cumulative count sent. On the right, a Spark job started with `spark-submit` runs the streaming consumer. Each micro-batch triggers "Processing batch" messages, and within each batch we print summary statistics of the predicted delay probabilities.

This screenshot is more than cosmetic. It demonstrates that the model is not merely an offline artifact; it is genuinely deployed inside a streaming context, reading and scoring data with latencies on the order of seconds. In practice we found that the micro-batch size and trigger interval could be tuned to trade off end-to-end latency against throughput.

VI. Machine Learning Methods and Results

A. Modeling Setup

Using the features described in Section III, we build MLlib pipelines consisting of three main stages: (1) indexing and one-hot encoding of categorical variables, (2) vector assembly into a single feature vector, and (3) training of a classifier. We consider three algorithms:

Logistic Regression (LR). A linear model that estimates

$$\Pr(y = 1 \mid \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x} - b)}.$$

We apply L2 regularization to prevent overfitting and use Spark's limited-memory BFGS optimizer.

Decision Tree (DT). A tree-based classifier that recursively splits the feature space to minimize impurity. Trees naturally capture interactions and nonlinearities but can overfit high-cardinality features.

Random Forest (RF). An ensemble of decision trees trained on bootstrap samples with feature subsampling [3]. By averaging predictions across trees, random forests reduce variance and often improve generalization.

We randomly split the dataset into 80% training and 20% test sets, stratified by label so that the proportion of delayed flights is similar in both. All hyperparameters (regularization strength for LR, maximum depth and number of bins for DT, and number of trees and maximum depth for RF) are tuned using cross-validation on the training set.

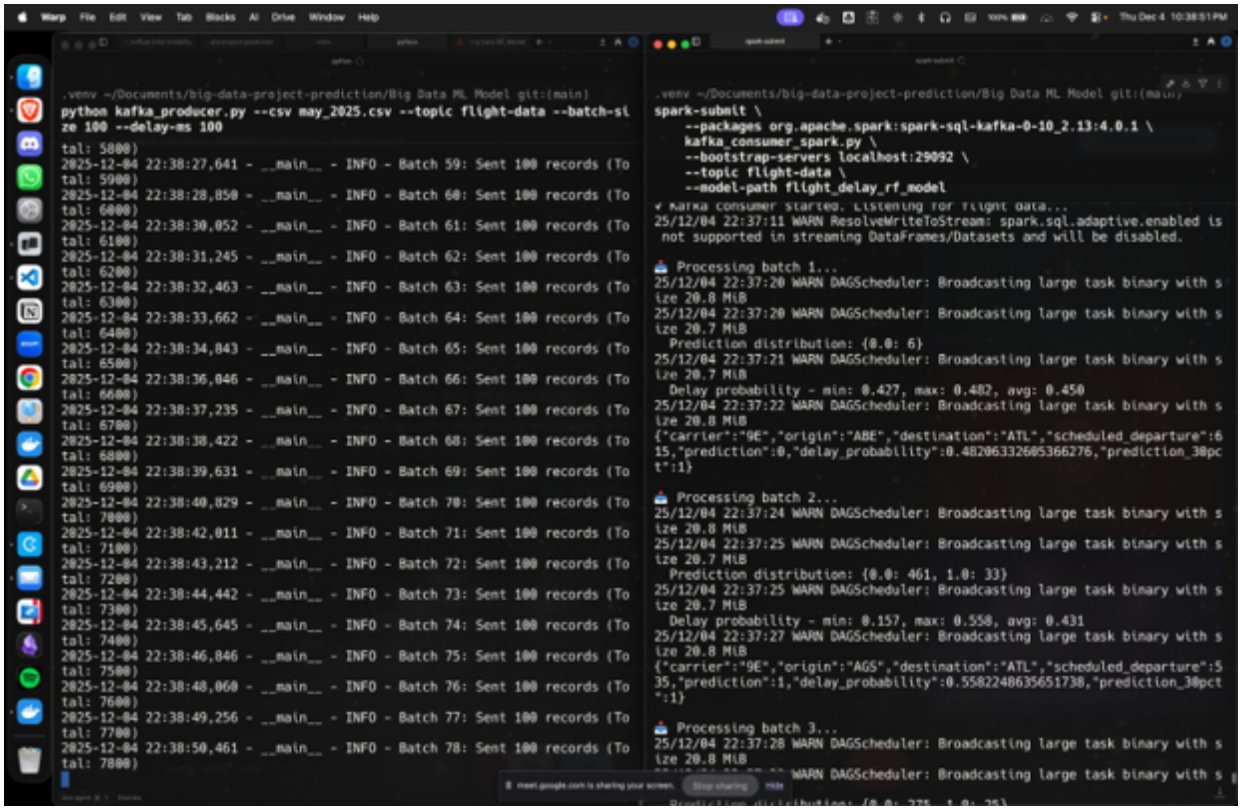

Fig. 7: Streaming experiment in progress. The left terminal shows the Kafka producer sending batched flight records, while the right terminal shows the Spark Structured Streaming job consuming those records and computing delay probabilities in micro-batches.

```
================================================================================
MODEL COMPARISON SUMMARY
================================================================================
              Model  Accuracy  Precision   Recall  F1-Score  AUC-ROC
Logistic Regression  0.787165   0.619628 0.787165  0.693420 0.625675
      Decision Tree  0.784733   0.734021 0.784733  0.732038 0.491521
      Random Forest  0.790624   0.756006 0.790624  0.711078 0.694309
```

Fig. 8: Model comparison summary. Random forest achieves slightly higher accuracy and F_1-score than logistic regression and decision tree, with the strongest AUC-ROC.

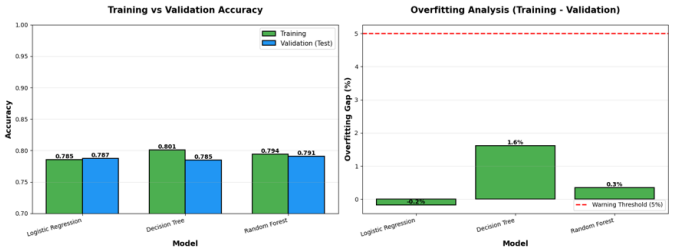

Fig. 9: Left: training vs. validation accuracy for each model. Right: overfitting gap (training minus validation) with a dashed line at 5% as a rough warning threshold.

B. Model Comparison

Figure 8 reproduces the ASCII table produced by our model comparison script. The table lists accuracy, precision, recall, F_1-score, and AUC-ROC for each model on the test set.

Logistic regression reaches an accuracy of about 0.787 and an F_1-score around 0.693. The decision tree performs similarly in accuracy but with slightly better F_1, suggesting it picks up some nonlinear patterns, although its AUC-ROC is lower, indicating less robust ranking of probabilities. The random forest outperforms both, achieving an accuracy around 0.791 and F_1 around 0.711 with the best AUC-ROC (about 0.694). These results support our choice of random forest as the primary model for deployment.

C. Overfitting Analysis

To verify that the random forest is not simply memorizing the training data, we compare training and test accuracies for all three models. Figure 9 shows two bar charts: one for training vs. validation accuracy, and another for the "overfitting gap" (training minus validation) expressed as a percentage.

The logistic regression bars almost overlap, with only a negligible gap slightly below zero due to sampling variation; the model generalizes as expected for a regularized linear classifier. The decision tree trains to higher accuracy than it attains on validation data, resulting in an overfitting gap of about 1.6%. While this is not catastrophic, it confirms that deeper trees start to fit noise or idiosyncratic combinations of features. The random forest sits between the two extremes: training accuracy is modestly higher than test accuracy, but the gap is around 0.3%, well below the 5% guideline line drawn in the figure. Ensemble averaging evidently mitigates overfitting while retaining the ability to model nonlinear structure.

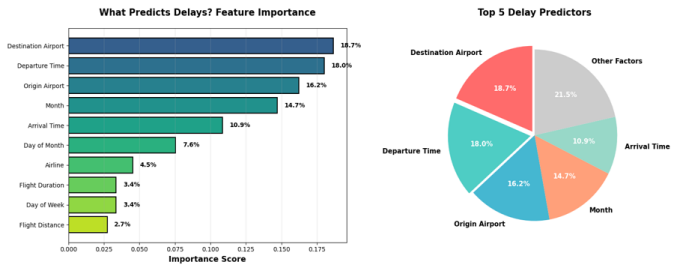

Fig. 10: Left: global feature importance scores from the random forest model. Right: distribution of importance mass for the top five predictors versus all remaining features.

D. Feature Importance

Random forests provide natural measures of feature importance based on how much each variable reduces impurity across the ensemble. Figure 10 presents two complementary visualizations. On the left, a horizontal bar chart shows importance scores for the top features; on the right, a pie chart aggregates the top five predictors and groups all remaining features into an "Other factors" slice.

Destination airport emerges as the single most important feature, with an importance around 18.7%. This reflects the role of heavily congested hubs and weather-prone airports: certain destinations regularly experience holding patterns and ground delay programs, leading to recurrent arrival issues. Departure time is almost as influential (18.0%), capturing the fact that flights departing in peak bank periods or late at night face different delay regimes. Origin airport appears next (16.2%), followed by month (14.7%) and arrival time (10.9%). Together these top five features account for more than three-quarters of the total importance mass.

The "Other factors" slice in the pie chart includes airline, distance band, day of week, and more subtle interaction indicators. Their individual contributions are smaller but not negligible. For example, airline importance partly overlaps with origin and destination because some carriers dominate certain hubs. Distance captures the limited ability of short-haul flights to recover from delays en route. The importance distribution reinforces the idea that any operational delay management strategy must consider where and when flights operate, not just which airline runs them.

VII. Streaming Demo and Visualization

A. Real-Time Prediction Loop

Once the random forest model is trained and validated, we serialize it using Spark MLlib's built-in model saving functions. The serialized model and the associated feature engineering pipeline are then loaded by a separate Spark Structured Streaming job. A Kafka producer script reads flight records from the May 2025 CSV file and emits them as JSON messages to a topic, throttling the rate so that the stream roughly resembles live traffic.

As shown in Figure 7, the streaming job processes records in micro-batches. For each batch, the job applies the same `StringIndexer`, `OneHotEncoder`, and `VectorAssembler` stages to incoming messages before passing them to the random forest. The job then computes summary statistics such as the proportion of flights predicted to be delayed and the distribution of predicted probabilities. These summaries are logged in the right-hand terminal and simultaneously sent to downstream sinks.

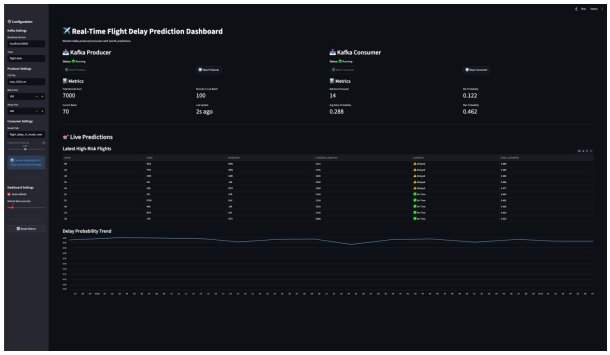

Fig. 11: Prototype dashboard for monitoring streaming delay predictions. Aggregated statistics and time-series plots allow users to see emerging disruption patterns in near real time.

B. Dashboard

To make the predictions interpretable for non-technical stakeholders, we created a simple dashboard. Figure 11 shows a screenshot: the top portion lists key performance indicators such as the number of flights processed in the last window, the current estimated fraction of high-risk flights, and breakdowns by airline or airport. The lower panel displays a time-series chart of the share of flights whose predicted delay probability exceeds a configurable threshold.

Although minimal compared to commercial airline control-center systems, the dashboard demonstrates the complete loop from historical data to operational insight. A dispatcher could, in principle, glance at the chart and identify when delay risk is starting to spike for a particular hub or airline, prompting further investigation or proactive passenger communication.

VIII. Discussion, Limitations, and Future Work

The project shows that widely available Big Data tools are sufficient to build a reasonably accurate and operationally meaningful flight delay predictor. The EDA phase revealed strong weekly and seasonal patterns, highlighted the central role of destination and origin airports, and quantified the relationships among different delay causes. Random forest models captured this structure and performed well under cross-validation. The streaming integration confirmed that the model could be embedded into a real-time scoring loop with modest latency.

At the same time, several limitations should be acknowledged. First, the model uses only BTS internal features; it does not explicitly incorporate weather forecasts, airport surface conditions, or air traffic control initiatives, all of which are known to influence delays. Some of this information is indirectly encoded in historical patterns, but reliance on historical averages can be misleading when encountering unusual storms or policy changes.

Second, we treated all routes and carriers uniformly. In reality, delays on certain routes are less harmful (for instance, leisure routes with few connections), while others have large knock-on effects. A richer system would weigh flights by their network impact rather than treating every instance equally in the loss function.

Third, our evaluation window is limited to a 15-month sample. A more robust assessment would examine multiple disjoint time periods and explicitly test how well a model trained on one year generalizes to the next, especially across unusual events such as pandemics or large regulatory changes.

Future work could follow several directions. On the modeling side, gradient-boosted trees or calibrated probabilistic classifiers might improve both accuracy and probability estimates. On the data side, integrating real-time weather feeds, airport configuration data, and information about ground delay programs would make predictions more situationally aware. On the systems side, exploring exactly-once semantics, model versioning, and automated retraining workflows would bring the architecture closer to what airlines might deploy in production.

IX. Conclusion

This project demonstrates a complete Big Data workflow built around a real, messy, and operationally important dataset. By combining Apache Spark, Spark SQL, Spark MLlib, Apache Kafka, and Spark Structured Streaming, we created a system that starts with multi-gigabyte historical BTS archives, learns patterns of delay, and then uses those patterns to score incoming flight events in near real time. The analysis confirms that when, where, and with which carrier a flight operates substantially affects its delay risk, and that relatively simple models, when trained on large datasets, can already provide useful foresight.

The work also illustrates broader lessons about Big Data engineering. Careful attention to schema consistency and feature engineering can matter as much as algorithm choice. EDA remains vital even at large scale, both to validate assumptions and to explain model behavior to stakeholders. Finally, unifying batch and streaming within a single design lowers the friction between exploratory analysis and deployment, making it easier to turn insights into action.

References

[1] M. Ball, C. Barnhart, M. Dresner, M. Hansen, K. Neels, A. Odoni, *et al.*, "Total delay impact study: A comprehensive assessment of the costs and impacts of flight delay in the United States," NEXTOR Report, Federal Aviation Administration, 2010.

[2] Bureau of Transportation Statistics, "Reporting Carrier On-Time Performance (1987–present) [Data set]," U.S. Department of Transportation, 2024. [Online]. Available: https://www.transtats.bts.gov

[3] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[4] D. W. Hosmer, S. Lemeshow, and R. X. Sturdivant, *Applied Logistic Regression*, 3rd ed. Hoboken, NJ, USA: Wiley, 2013.

[5] J. J. Rebollo and H. Balakrishnan, "Characterization and prediction of air traffic delays," *Transportation Research Part C: Emerging Technologies*, vol. 44, pp. 231–241, 2014.

[6] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. Hong, M. Konwinski, *et al.*, "Apache Spark: A unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[7] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," in *Proc. NetDB*, 2011.

AI Use Acknowledgement

Portions of the wording in this report were drafted and revised with the assistance of an AI-based language tool. The authors reviewed, edited, and approved all content and are solely responsible for the data analysis, code, modeling decisions, and interpretations presented here.