Phase 2: Team Oracle Write Up

By: Matthew Love, Suraj Rajendran, Prathic Sundararajan

Provide a high level summary of your approach: what did you do and why?
Our primary strategy involved the development of a unique ensemble model, consisting of a Federated Artificial Neural Network (FANN) and our original XGBoost algorithm, which was exclusively trained on public data. This decision stemmed from our intention to leverage and optimize the distinct features and strengths of both methodologies. The FANN was implemented using a federated approach and trained on a comprehensive dataset encompassing both public and private information. The federated learning approach was selected to ensure robustness and increased accuracy of the model while maintaining the privacy of sensitive data.

In order to obtain the final predictive outcome, we employed a weighted average method between the FANN and XGBoost model's predictions. This was based on the consideration of not only retaining the reliability of individual models but also introducing a layer of collaborative intelligence. The integration of our previously developed XGBoost model was aimed at harnessing the superior performance of tree-based models on tabular data. The use of the ensemble method allowed us to benefit from the inherent strengths of the XGBoost model while concurrently improving on its potential weaknesses. The incorporation of the ANN in our approach served a dual purpose: it allowed us to integrate and process a wider range of data sources (including public and private datasets), and offered us the flexibility to capture non-linear patterns, which can often be a limitation with tree-based models. This combined approach enabled us to optimize predictive performance by harnessing the power of diverse machine learning architectures, while also addressing privacy concerns inherent to the use of sensitive private data.

What ways did you consider adapting your Phase 1 approach to accommodate federated learning? Which did you eventually implement and why?
In the initial phase of adapting our Phase 1 methodology to federated learning, we incorporated some of the same features that had been utilized in the creation of the public datasets. This approach aimed to maintain consistency and compatibility between the phases, thus leveraging any underlying patterns previously identified. Our first consideration was to apply a federated version of the XGBoost model, aligning with the approach used in Phase 1. However, upon further research and analysis, we discovered that the performance of XGBoost could potentially deteriorate in a federated context. This led us to reevaluate the use of this model for our federated learning strategy.

When establishing the ANN model, we endeavored to generate new features, particularly leveraging standtimes, arrival times, and airline codes. We created a suite of aggregated features using this information tailored to our airline-specific private dataset. This innovative approach allowed us to extract more valuable insights from these unique, industry-specific parameters. We ventured into experimental deep learning architectures for the ANN, conducting numerous trials with varying setups. Eventually, we settled on a model that utilized batch normalization along with four dense layers. The decision to use this architecture was driven by its performance during testing, where it demonstrated enhanced efficiency and robustness compared to alternative models. We implemented an ensemble technique with the objective of reducing the potential noise from the predictions of the federated ANN. ANN models can be particularly sensitive to noisy data, and the incorporation of the ensemble technique served to mitigate this vulnerability, enhancing the overall accuracy and reliability of our predictions.

Compare the performance (in terms of mean absolute error or other relevant metrics) of your Phase 1 and Phase 2 models.
We will first analyze the performance of our XGBoost trained on public features to our model in Phase 1. Despite only using public features and utilizing no internal boosting (like we did in Phase 1), we don't see significant drops in performance. While slightly higher, the MAE across different airports is still close to the values we reported in Phase 1 during internal testing. The average MAE across airports is 12.02, and a standard deviation of 2.24. The ANN has an MAE of 23.05 with a standard deviation of 2.15, which is significantly worse. This might be attributable to the ANN architecture or performance decreases that come about due to the federated nature of the model.

What experiments did you undertake to optimize performance? Do you believe your final Phase 2 submission achieves the best possible performance of a federated version of your Phase 1 model? If not, what do you think would be needed to achieve the best possible performance? What are the trade-offs involved between performance and privacy for your Phase 2 solution?
To optimize the performance of our Artificial Neural Network (ANN), we conducted a series of experiments focusing on hyperparameter tuning. By varying these parameters, we were able to fine-tune our model, resulting in improved prediction accuracy and generalization capabilities. Furthermore, we explored the effects of varying degrees of downsampling within our dataset. This experimental process enabled us to strike a balance between handling an extensive dataset and ensuring a manageable computational load, whilst maintaining model performance.

We discovered that the strategy of ensembling our ANN with the XGBoost model significantly boosted overall performance. This technique provided a holistic solution that maximized the strengths of both the ANN and XGBoost models while minimizing their individual weaknesses. Upon analysis of the private features in the dataset, we concluded that they did not contribute significantly to the prediction of minutes to pushback. Most of the predictive power appeared to be derived from the public features, which were critical to both the ANN and XGBoost models. Consequently, we observed a minor performance drop between Phase 1 and Phase 2, attributed to the inclusion of less informative private features. However, we believe that the performance of our Phase 2 solution remains robust and is a successful adaptation of the Phase 1 model to a federated context.

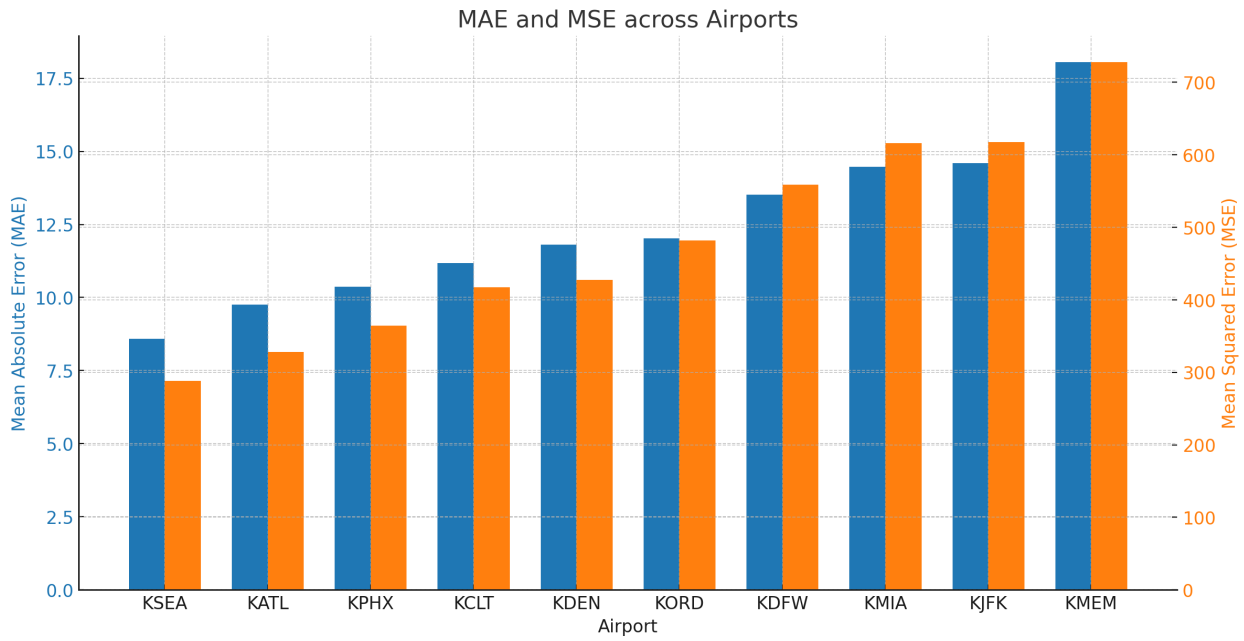
In terms of the trade-offs between performance and privacy, the most significant compromise came with the inclusion of the federated ANN. While this approach helped integrate private data securely, the resulting predictions were somewhat noisier, affecting the overall accuracy of the model. Despite this, the trade-off allowed us to uphold privacy standards, a critical factor in the context of federated learning.

What do you see as the most significant remaining work needed to deploy your Phase 2 solution in the real world? For example, coordinating training and/or evaluation, incorporating other private variables, addressing privacy concerns, etc.
To enhance the predictive power of our Phase 2 solution for real-world deployment, it would be advantageous to consider incorporating additional private features across airlines. These potential features, though currently not accessible to us, could further enrich the dataset and improve our model's performance while still preserving privacy under the federated learning framework.

Ideally, we envision an advanced federated model that is easily adaptable to specific airports, offering a blend of generalization and personalization. This could manifest as a model (for instance, an ANN) where most layers are common across all airlines, providing generalized learning, while the last few layers are airline-specific, offering bespoke insights. This approach would capitalize on the shared features across airlines while accounting for individual airline nuances, resulting in enhanced prediction accuracy. Despite the benefits of federated learning and our adaptations, the XGBoost model trained on public data remains a crucial component of our system. It provides a reliable baseline for predictions, substantiating the accuracy of our ensemble model, and facilitating the robustness of our overall prediction capability.

In terms of data management, our current use of the Flower API for data creation might not be the most efficient option for real-world deployment. It would be worthwhile to explore more streamlined and efficient methods of data creation and management to improve the speed and efficiency of model training and updating. To ensure the successful deployment of our solution, it will also be essential to address privacy concerns robustly. This will include stringent data management protocols and effective anonymization techniques to uphold data confidentiality and maintain trust with airlines and regulatory bodies.

Do you have any useful charts, graphs, or visualizations from the process?



This is the MAE and MSE across all airports for our solution.

Copy and paste the 3 most impactful parts of code you developed during Phase 2. Explain what each does and how it helped your model.

```python
# Loading in Standtimes
standtimes = pd.read_csv(
    #f"{directory}{airport}/private/{airport}/{airport}_{airline}_standtimes.csv",
    data_directory / airport / "private" / airport / f"{airport}_{airline}_standtimes.csv",
    parse_dates=["timestamp", "arrival_stand_actual_time", "departure_stand_actual_time"]
)
standtimes[['plane_id','departing_airport_code','arriving_airport_code']] =  standtimes.gufi.str.split('.', expand = True)[[0,1,2]]
standtimes['airline_code'] = standtimes.gufi.str[:3]

timestamps = pd.to_datetime(features.index.get_level_values("timestamp"))

# We will keep track of the feature updates in a dictionary
# The dictionary keys are timestamps and the values are lists of tuples. Each tuple contains a gufi and the corresponding aircraft count
feature_updates = defaultdict(list)

for i, now in enumerate(timestamps):

    # Getting ETD Features
    now_features = features.xs(now, level="timestamp", drop_level=False)
    now_etd = etd_airline.loc[(etd_airline.timestamp > now - timedelta(hours=30)) & (etd_airline.timestamp <= now)]
    now_gufis = pd.Series(now_etd.gufi.unique())
    # Get the aircraft type for this row
    aircraft_type = features.iloc[i]['aircraft_type']
    aircraft_count = count_same_aircraft_type(now_gufis, mfs, aircraft_type)

    # Getting stand times features
    destination = features.index[i][0].split('.')[2]
    standtimes_dest = standtimes[(standtimes.timestamp > now - timedelta(hours=30)) & (standtimes.timestamp <= now)]
    standtimes_dest = standtimes_dest[standtimes_dest['arriving_airport_code'] == destination]
    times_to_dest = standtimes_dest['arrival_stand_actual_time'] - standtimes_dest['departure_stand_actual_time']
    mean_time_to_dest = times_to_dest.mean().total_seconds() / 60
    std_time_to_dest = times_to_dest.std().total_seconds() / 60
    median_time_to_dest = times_to_dest.median().total_seconds() / 60

    feature_updates[now].append((features.index[i], aircraft_count, mean_time_to_dest,
                                 std_time_to_dest, median_time_to_dest))

    print("Done with " + str(i+1) + " out of " + str(features.shape[0]) + " items", end='\r')
```

In the above code, we are generating a new private feature using a combination of stand times. There might be an assumption that certain destinations may be associated with greater push back times. To see if there is a relation here, we look at all the flights in the past 30 hours that went to the same destination as the current plane we are concerned with. We get all those planes' time to destination, as well as aggregate statistics of them.

```python
def initialize_model(n_features: int = 122):
    """
    Testing simple model to start for integration. Once there is a better understanding with
    others about features and where loading happens can apply changes.
    """
    logger.debug(f"MODEL HAS BEEN initialized")
    input_shape = (n_features,)
    input_layer = layers.Input(shape=input_shape)
    batch_norm = layers.BatchNormalization()(input_layer)
    dense0 = layers.Dense(n_features*2, activation='relu')(batch_norm)
    dense1 = layers.Dense(n_features*2, activation='relu')(dense0)
    dense2 = layers.Dense(n_features*2, activation='relu')(dense1)
    output_layer = layers.Dense(1, activation='relu')(dense2)

    model = keras.Model(inputs = input_layer, outputs = output_layer)

    model.compile(optimizer='adam', loss='mse', metrics=['mae'])

    return model
```

This is the ANN model we finalized on after performing multiple levels of hyperparameter tuning. It consists of four dense layers with relu activation. We chose relu for the final activation layer as we wanted to force outputs to be greater than 0.

```
presentDate = datetime.datetime.now()
save_dir_name = str(int(presentDate.timestamp()))
save_path = f'xgb_models/{save_dir_name}'
os.mkdir(save_path)
for airport in AIRPORTS:
    fn = f'{public_features_directory}/processed_public_features_{airport}.pkl'
    print(airport)
    with open(fn, 'rb') as f:
    # Load the object from the pickle file
        curr_data = pickle.load(f)
    curr_data["X"].index
    train_data = curr_data["X"].reset_index()
    # extract year, month, day, and hour information
    train_data['unix_time'] = train_data['timestamp'].astype(np.int64)  // 10**9
    train_data.dropna(inplace=True)
    train_data.reset_index(drop=True, inplace=True)
    label_data = curr_data["y"]
    merged_df = pd.merge(train_data, label_data, how = "left", on = ['timestamp', 'gufi'])
    feature_cols = merged_df.columns
    feature_cols = list(set(feature_cols) - set(['gufi', 'minutes_until_pushback','timestamp']))
    label_col = ['minutes_until_pushback']
    gss = GroupShuffleSplit(n_splits=1, train_size=0.3, test_size=0.7, random_state=42)
    # split the data into train and test sets
    train_index, test_index = next(gss.split(merged_df, groups=merged_df['gufi']))
    df_train = merged_df.iloc[train_index].copy()
    df_test = merged_df.iloc[test_index].copy()
    X_internal_train = df_train[list(set(feature_cols))]
    y_internal_train = df_train[label_col]
    X_internal_test = df_test[list(set(feature_cols))]
    y_internal_test = df_test[label_col]
    internal_regressor = xgb.XGBRegressor()
    print(f"Internal Regressor Len of Training: {len(X_internal_train)}")
    internal_regressor.fit(X_internal_train, y_internal_train)
    print(f"Internal Regressor Len of Prediction: {len(X_internal_test)}")
    y_internal_pred = np.int32(np.around(internal_regressor.predict(X_internal_test),decimals=0))
    internal_mae = mean_absolute_error(y_internal_test, y_internal_pred)
    internal_mse = mean_squared_error(y_internal_test, y_internal_pred)
    # # Print the results
    print(f"MAE: {internal_mae} & MSE: {internal_mse}")
    df_test['y_pred'] = y_internal_pred
    model_file_name = f"{save_path}/{airport}_xgb.pkl"
    pickle.dump(internal_regressor, open(model_file_name, "wb"))
```

In this section of code, we train our XGBoost model only on public data. We consider these models (airport specific) as a baseline measure of what the predicted time to pushback will be. Given the model performed well only with public level features in Phase 1, we believe that its inclusion in this phase will provide added advantage in performing accurate predictions.

Please provide the machine specs and time you used to run your model.
CPU (model): Apple M1 Max
GPU (model or N/A): N/A
Memory (GB): 64 GBs
OS: MacOS
Train duration: 12 hours (mostly feature engineering)
Inference duration: 1 hour

Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?
As you work with our model, we recommend monitoring memory usage and ensuring that your system can handle the computational requirements. Additionally, it is crucial to keep an eye on any potential numerical stability issues that may arise during the model's operation. Especially during data processing, you may face memory issues if no downsampling is performed. During model training, we have set the batch size to 16 to ensure that our computing systems are able to train the model without running into errors. However, if your system permits, increasing the batch size should be okay.

Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?
During the initial stages of our project, we utilized additional resources to gain a deeper understanding of the problem domain and the data provided. To supplement our analysis, we explored online flight trackers to familiarize ourselves with real-world flight operations and air traffic management. This helped us gain insights into the underlying patterns and relationships in the data that might not be readily apparent from the dataset alone. Furthermore, we conducted a literature search to review state-of-the-art methods in federated learning as well as personalized federated learning. We investigated the federated XGBoost method (FedBoost) but decided not to proceed with it. Although these resources were not directly included in our code submission, they played a vital role in shaping our understanding of the problem and informing our approach to developing an effective model for predicting pushback times.

How did you evaluate performance of the model other than the provided metric, if at all?
Same as Phase 1
Apart from the provided metric, we conducted an experiment to evaluate our model's performance under different data conditions. We intentionally reduced the amount of training data to only 30% of the original dataset to observe the impact on our model's performance. This experiment aimed to assess the model's robustness, stability, and generalizability when confronted with limited data availability. We also evaluated our models using MSE, in addition to MAE as it is a differentiable function. This allows us to better train our deep learning models.

By analyzing the performance of our model under such constraints, we were able to gain insights into the model's potential weaknesses and strengths. This additional evaluation technique helped us better understand our model's behavior and potential limitations, informing possible improvements and adjustments to enhance its performance and adaptability in various data scenarios.

What are some other things you tried that didn't necessarily make it into the final workflow? (quick overview)

The other things we tried fall broadly into two groups, feature generation and federated model architecture. We tried many feature generation methods, specifically combining the private and public data but none provided significant predictive performance. We also experimented with several federated architectures but we dropped them due to time, computational, or performance issues.

Are there any other learnings or conclusions from Phase 2 that are not covered in the previous questions that you would like to include? (optional)
N/A