Prize recipient solution documentation guide

This document details the information required to submit your Phase 2 solution materials.

I. **Code submission and result reproducibility**
You package up and send us your code, documentation of dependencies, and any other assets you used. We review your package and make sure that it works and that we can fully reproduce the workflow from raw data to a submission comparable to your best submission. See the Competition prize winner submission template for more guidance on packing up and organizing your code (however, please note that the template's write-up questions differ from the questions for this challenge).

II. **Model documentation and write-up**
You write up answers to our questionnaire, providing important context and documentation so that the beneficiary and the community get the most out of your work.

I. Code submission and result reproducibility

You will need to submit a compressed archive of your code. You don't need to include the raw data we provided, but everything else should be included and organized. If the files are too large to be e-mailed, a Google Drive or Dropbox share (or other comparable method of transferring data) works.

Note: *Please follow these instructions carefully*. The spirit and purpose of the competition (and the reason for offering prizes) is to give our beneficiary organizations the best possible solution *along with working code they can actually use*. In accordance with the competition rules, if we can't get your code working and reproduce your results with a reasonable effort, or if your entry is too disorganized to be practically usable, then your entry may be disqualified!

The overall concept is to **set up this archive as if it were a finished open source project**, with clear instructions, dependencies and requirements identified, and code structured logically with an obvious point of entry. Check out the competition prize winner README template to get started. We also have a data science project template which may be helpful.

At a minimum, your solution must include **an extremely clear README** that details all of the steps necessary to get to your submission from a fresh system with no dependencies (e.g., a brand new Linux, Mac OS X, or Windows installation depending on what environment you choose to develop under) and no other data aside from the raw data you downloaded from us.

This will probably entail the following:

- Necessary tools and requirements (e.g. "You must install Word2Vec 0.1c" or "Install the required Python packages in `requirements.txt`").
    - **All requirements should be clearly documented**, for instance in either a requirements.txt file with versions specified or environment.yml file.
- The series of commands, in order, that would get a reasonably experienced and savvy user from your code to a finished submission.
    - **Ideally, you will have a main point of entry to your code** such as an executable script that runs all steps of the pipeline in a deterministic fashion. A well-constructed Jupyter notebook or R script meets this standard.
    - **The next best thing is a list of specific, manual steps** outlining what to do. For example, "First, open Word2Vec and set these parameters. [...] Take the output file and run the script `src/make_preds.R` with the following parameters [...]". *(The limitations of this approach should be clear to any experienced data scientist!)*
- **Make sure to provide access to all trained model weights necessary to generate predictions from new data samples** without needing to retrain your model from scratch. Note that model weights can be contained in your archive or shared via a cloud storage service. The solution should provide clear instructions to perform inference on a new data point, whether or not it is included in the test set.
- Any other instructions necessary to end up with your winning submission file (or comparable — we understand that certain parts of model fitting are stochastic and won't result in exactly the same parameters every time).

II. Model documentation and write-up

Information included in this section may be shared publicly with challenge results. You can respond to these questions in an e-mail or as an attached file. Please number your responses.

1. Provide a high level summary of your approach: what did you do and why?

   For the centralized solution in Phase 1, we used LightGBM models trained on 30+ features. For Phase 2, we divided these features in three categories:
   - **Private features:** aircraft_type, major_carrier, flight_type
     These are features that are directly available to each airline and that are no longer publicly shared in Phase 2.
   - **Computed private features:** arr_taxi_3hr [not used in Phase 2], arr_taxi_30hr [not used in Phase 2], arrs_taxiing [not used in Phase 2]
     These are features that we extracted by aggregating arrival_stand_actual_time values across different airlines in Phase 1.
   - **Public features:** all other features.

   For Phase 2 we developed two approaches:
   1. **Centralized_without_private_information solution (LightGBM with public features):** we eliminated the “private features” and “computed private features” from our feature set and retrained LightGBM models on the public features only. This is not a form of federated learning and does not involve use of the Flower framework.
   2. **Federated solution (MLP with private and public features):** we trained a neural network model (multilayer perceptron, MLP) in Flower over the public features and the private features. The MLP architecture consists of a normalization layer, followed by three dense ReLU layers with 32, 64, and 64 neurons respectively, and a linear layer with 1 output neuron. An important note is that, already in our Phase 1 solution, we encoded the private features using ordinal encoding. To keep the encoding consistent across all clients, we assumed that the list of possible values of the private features is publicly available. We used this list to construct an encoder for each private feature, and then pushed those encoders to the clients for use.

   We did not use the “computed private features” in our Phase 2 solution because these arrival-specific features would need to be extracted in a privacy-preserving manner across the federation, i.e. through aggregating raw data held by all the clients in the federation. While such privacy-preserving extraction could be performed using e.g. secure multi-party computation protocols, we decided not to pursue this route because we observed that the contribution of the computed private features to the overall utility of the models was too small to justify such an effort.

   We have not used the aircraft_engine_class feature in our Phase 1 and Phase 2 solution because we observed that it had little or no effect on the utility. Including this feature in our federated solution would be easy, and could be done in exactly the same way in which we included the other “private features” for training and inference.

2. What ways did you consider adapting your Phase 1 approach to accommodate federated learning? Which did you eventually implement and why?

   For the centralized solution in Phase 1, we used LightGBM models trained on 30+ features. When converting this solution into a federated solution in Flower for Phase 2, we encountered two main issues:
   1. The Flower framework for federated learning provides support for training of neural networks. Support for training of gradient boosted decision trees in Flower is emerging and not as well developed yet as training of neural networks.
   2. Some of the features that we used in Phase 1 are computed by aggregating private arrival_stand_actual_time values across different airlines. This means that the use of these features would require a privacy-preserving feature extraction step in the federation before they are available to clients for use.

   To address these issues we:
   1. Changed from LightGBM models to MLP models in our “Federated solution” for Phase 2.
   2. Omitted the “computed private features” in Phase 2. Ultimately, these features contributed too little to the overall utility to justify the development and use of secure multi-party computation protocols for their extraction.

3. Compare the performance (in terms of mean absolute error or other relevant metrics) of your Phase 1 and Phase 2 models.

   Below we compare three approaches in terms of **MAE on the Phase 2 data**.
   - Phase1_C: Centralized solution:
     LightGBM with private features, computed private features, and public features
   - Phase2_C: Centralized_without_private_information solution:
     LightGBM with public features
   - Phase2_F: Federated solution:
     MLP with private features and public features [global model]
   - Phase2_C: Centralized solution:
     MLP with public features [global model]

	Phase1_C	Phase2_C	Phase2_C	Phase2_F
	LightGBM	LightGBM	MLP	MLP
public features private features computed private features	yes yes yes	yes no no	yes no no	yes yes no

KATL	7.8237	7.8745	-	-
KCLT	9.3225	9.3217	-	-
KDEN	11.6925	11.7278	-	-
KDFW	12.1088	12.1598	-	-
KJFK	10.9442	11.0474	-	-
KMEM	17.9373	18.2305	-	-
KMIA	12.178	12.2101	-	-
KORD	11.1276	11.2117	-	-
KPHX	9.4983	9.5362	-	-
KSEA	7.2366	7.2485	-	-
All	10.8373	10.9093		17.3736

Airlines	Phase2_C, MLP	Phase2_F, MLP
AAL		21.3
ASA		12.9
ASH		20.0
AWI		17.0
DAL		14.3
EDV		17.5
EJA		14.3
ENY		29.9
FDX		16.2
FFT		24.8
GJS		14.6
JBU		18.3

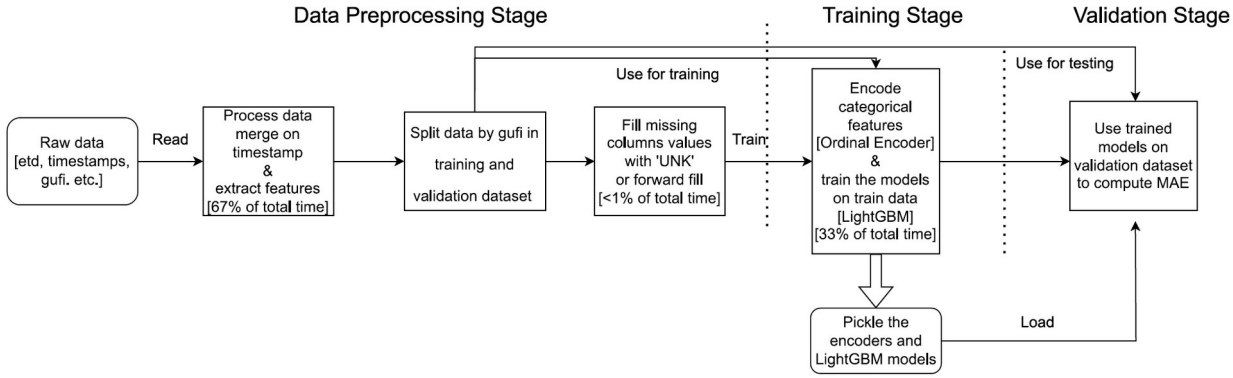
JIA		14.9
NKS		15.9
PDT		20.1
QXE		18.6
RPA		17.0
SKW		16.6
SWA		14.3
UAL		14.7
UPS		11.4
ALL		17.3736

4. 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?
   We experimented with various architectures and models, however, we chose to invest the majority of the available time into the MLP model, since it is the type of models supported and documented in Flower most extensively. After empirically obtaining the train/validation accuracies for various possible model architectures, we determined that 3-layer MLP appears to be complex enough to have a sufficiently low overall validation MAE while simple enough to prevent unnecessary overfitting, minimizing the delta between train and validation accuracy. While certain potential improvements could be implemented through fine tuning individual models' hyperparameters and architecture to fit the individual airline best, taking in account the amount of data and trends in the airlines, we believe that the current iteration of the MLP possesses the accuracy which is generally comparable to that of our final Phase 1 LightGBM model. In order to preserve the privacy of data, we decided to not utilize the computed features, due to potential data leakage at the moment of assembling the computed features with private and public features, which may slightly increase the overall MAE.

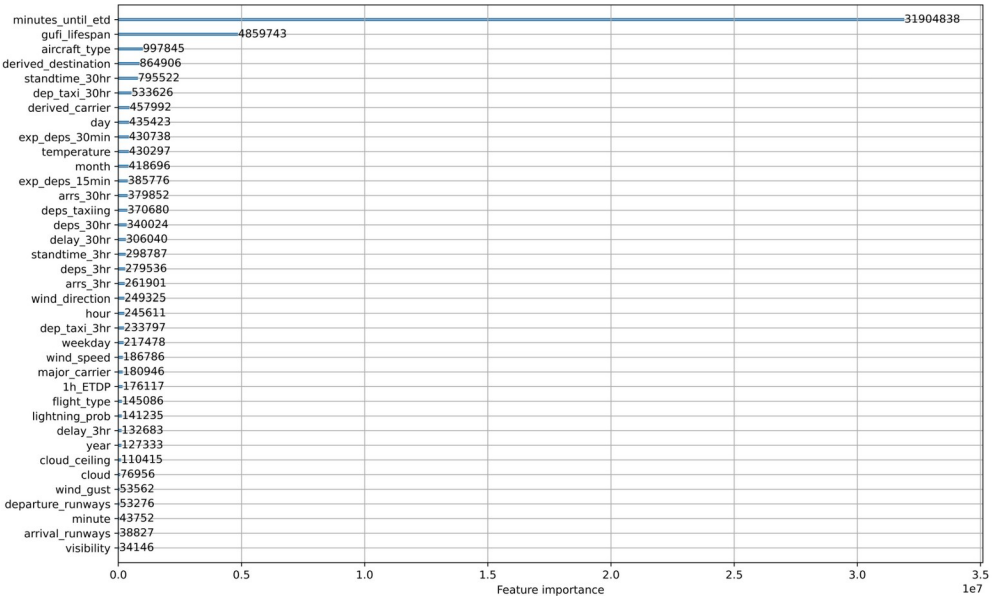
5. What do you see as the most significant remaining work needed to deploy your Phase 2 solution in the real world? For example, coordinating training and/or evaluation, incorporating other private variables, addressing privacy concerns, etc.
   We suppose that if more data would be private, which is possible in a real-world application of the federated learning models, the possibility of securely including the computed features will be among the primary questions. As was mentioned earlier, the question of implementing

differential privacy and encryption for the provided setup was the one requiring a significant amount of effort for us to resolve properly.

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



Overall Process Diagram



Phase 1 feature Importance Graph, LightGBM

7. 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.

Serve Side Evaluation

```
def get_evaluate_fn(model, test_loaders):
    # Use the validation set to have the default values
```

```
    def evaluate(
        server_round: int, parameters: NDArrays, config: Dict[str, Scalar]
    ) -> Optional[Tuple[float, Dict[str, Scalar]]]:
        model=Net().to(DEVICE)
        set_parameters(model, parameters)  # Update model with the latest parameters
        losses=0
        accuracies=0

        for i in range(len(test_loaders)):
            losses = losses + test(model, test_loaders[i])

        loss = losses/len(test_loaders)
        print(f"Server-side evaluation loss / accuracy {loss}")

        return float(loss), {"accuracy": float(loss)}

    return evaluate
```

Allowed team to have an objective validation evaluation standard to compare various federated strategies and model configurations on a standardized validation dataset, allowing to simulate the centralized inference, leading to a more proper prediction process that could be submitted for the organizers.

Model Configuration

```
class Net(nn.Module):
    def __init__(self) -> None:
        super(Net, self).__init__()
        self.fc1 = nn.Linear(len(features), 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, 1)


    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Our team found that to be a largely effective model architecture, leading to a relatively low overall loss and optimal training time, effective for relatively rapid prototyping, allowing to potentially experiment with various federated strategies and optimize the final performance for lower overall MAE.

Training and testing dataset separation

```
# get the train and test dataset
def get_train_and_test_ds(_airport: str, _airline: str = "PUBLIC") -> tuple[pd.DataFrame,
pd.DataFrame]:
    # load data
    train_df: pd.DataFrame
    val_df: pd.DataFrame
    if _airline != "PRIVATE_ALL":
        train_df = get_train_tables(_airport, _airline, remove_duplicate_gufi=False)
        val_df = get_validation_tables(_airport, _airline, remove_duplicate_gufi=False)
    elif _airport.upper() != "ALL":
        train_df = pd.concat(
            [
                get_train_tables(_airport, each_airline, remove_duplicate_gufi=False)
                for each_airline in AIRLINES
                if os.path.exists(get_train_tables_path(_airport, each_airline))
            ]
        )
        val_df = pd.concat(
            [
                get_validation_tables(_airport, each_airline, remove_duplicate_gufi=False)
                for each_airline in AIRLINES
                if os.path.exists(get_validation_tables_path(_airport, each_airline))
            ]
        )
    else:
        train_df = get_all_train_tables("ALL", remove_duplicate_gufi=False)
        val_df = get_all_validation_tables("ALL", remove_duplicate_gufi=False)
    # load encoder
    _ENCODER: dict[str, OrdinalEncoder | OneHotEncoder] = get_encoder()
    # need to make provisions for handling unknown values
    for col in ENCODED_STR_COLUMNS:
        if col in train_df.columns:
            if isinstance(_ENCODER[col], OrdinalEncoder):
                train_df[[col]] = _ENCODER[col].transform(train_df[[col]])
                val_df[[col]] = _ENCODER[col].transform(val_df[[col]])
            else:
                train_df = pd.concat([train_df, _ENCODER[col].transform(train_df[[col]])],
axis=1).drop([col], axis=1)
                val_df = pd.concat([val_df, _ENCODER[col].transform(val_df[[col]])],
axis=1).drop([col], axis=1)
    for col in ENCODED_STR_COLUMNS + CATEGORICAL_INT8_COLUMNS:
        if col in train_df.columns:
            train_df[col] = train_df[col].astype("int8")
```

```
            val_df[col] = val_df[col].astype("int8")
    # drop useless columns
    train_df.drop(columns=(_col for _col in _FEATURES_IGNORE if _col in train_df),
inplace=True)
    val_df.drop(columns=(_col for _col in _FEATURES_IGNORE if _col in val_df),
inplace=True)

    return train_df, val_df
```

Provided a configurable function that combined the provided Phase 2 airline csv files into corresponding private and public datasets separated by individual airlines that will be utilized for training and testing federated models before the final submission environment, simplifying the overall code.

8. Please provide the machine specs and time you used to run your model.
   - CPU (model): AMD Ryzen 9 7950X 16-Core Processor
   - GPU (model or N/A): 4090
   - Memory (GB): 65.5 GB
   - OS: Linux, Ubuntu
   - Train duration: approximately 7 hours
   - Inference duration: approximately 20 minutes

9. Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?
   Large memory requirements, similarly to our initial Phase 1 LightGBM model, is a potential concern for the users without sufficient hardware available, and the overall long span of time required for training and inferring the results may be a potential limitation. Also, a certain degree of knowledge about the possible categorical feature values was assumed, in order to produce a constant ordinal categorical feature encoding, such that we assumed that each categorical feature assumes certain range of values, and therefore each possible value, inferred from Phase 1, was assigned a corresponding integer number, such that training and testing would be standardized among all of the airline clients.

10. Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?
    Our team utilized the lightGBM native feature importance graph through plot_importance method for finding which particular features were contributing to the change in validation MAE the most through ordering them by testing information gain.

11. How did you evaluate performance of the model other than the provided metric, if at all?

While evaluating the effectiveness of individual features was done through analyzing the information gain produced by each feature in addition to recording the associated change in the validation MAE in a table from the previous set of used features, the model as a whole was evaluated through computing the MAE and keeping the model trained on the feature set with lowest overall MAE, known as l-1 loss.

12. What are some other things you tried that didn't necessarily make it into the final workflow? (quick overview)
    We attempted to construct the models utilizing both TensorFlow and PyTorch library implementations. However, we ultimately chose to go with only one of them in our evaluation and training due to the minimal variances in the displayed validation MAE values between the two architectures.

13. 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)

    Main take-aways:
    - We found that simply removing the information that was made private in Phase 2 from our LightGBM model only slightly increased the overall MAE (from 10.8373 to 10.9093). Therefore, we suspect that the information that was made private in Phase 2 is not very valuable in the context of all the other information that was left publicly available. Given the data we were provided, it was more logical to train a centralized model solely on the publicly available data, as opposed to training a model that combines public and private data in a federated manner. This narrative is subject to alteration based on the specific type of data that is exclusively accessible in a private manner.
    - Flower mainly supports the federated learning of neural networks. Currently, there is limited support available for training other types of ML model architectures.
    - It appears that the privacy-preserving feature of extraction across clients is not natively supported in Flower. This could be achieved using a secure multiparty computation protocol (for input privacy) and/or using differential strategies (for output privacy) in a modular fashion, such as by encoding data in a preprocessing step.
    - It was assumed that all clients within the federation possess the identical ordinal encoder, thereby ensuring the consistent mapping of categorical private feature values to numbers (i.e., the identical categorical value is mapped to the identical number across all airlines). To achieve this, we assumed that such a mapping could be constructed globally and then given to all clients in the federation. We assumed that this would not result in any privacy loss, since the range of possible feature values for private features is public knowledge.