Northrop Grumman - Quantum AI at the Edge - Use Case Model

1. Functional Objectives

1.1 High Priority

- **For each of the standard models, should**:
  - Implement appropriate models (SVM, MLP)
  - Train the models on a supplied data set
  - Evaluate the performance of the models on a separate testing data set (accuracy, time to train, etc.)
  - Predict/classify new, unlabeled data

1.2 Medium Priority

- **For each of the quantum, and embedded models, should**:
  - Implement appropriate models (SVM, MLP)
  - Train the models on a supplied data set
  - Evaluate the performance of the models on a separate testing data set (accuracy, time to train, etc.)
  - Predict/classify new, unlabeled data
- **User should have means of measuring and comparing performance of each model type**

1.3 Low Priority

- **Reimplementing Quantum models on the emulator should show no significant performance improvements over traditional models.**

2. Non-Functional Objectives

2.1 Understandability

- System has the ability to display predictions in a format that is easy to interpret by the user.

2.2 Accuracy

- Each classifier is sufficiently accurate.

2.3 Reliability/Replicability

- Each individual model produces similar results in a similar amount of time when run several times on the same data.

2.4 Flexibility

- Methods used should be able to be extrapolated to new data sets
- The user has the option to supply hyperparameters for both underlying models as well as outputs.

2.5 Usability

- Users with ML knowledge should easily be able to interface with the created models and tailor them to their specific needs
- Users should be able to use trained classifiers without needing knowledge on how they work.

3. Use Cases

3.1 System Use Case Diagram

- **TBD (To Be Determined)**

3.2 Use Case Descriptions (for selected cases)

Train - TBD

Use Case Name:	Train
Summary:	Train is a method offered by a model which allows the user to pass in a set of labeled (training) data points to be used in training model parameters.
Basic Flow:	1. User selects which model to train 2. User finds and supplies training data (in matrix/tensor format) 3. User selects additional parameter values (e.g. learning rate, dropout rate, etc.) 4. Model trains using supplied data and parameters.
Alternative Flows:	If the model fails to converge, fails to run, or fails to converge within the time constraint, then we reevaluate the training dataset and start the process again from step 2.
Extension Points:	Predict, Evaluate
Preconditions:	Cleaning, munging, and wrangling the training dataset.
Postconditions:	A trained model that returns a predicted class label for a dataset

Predict - Completed

Use Case Name:	Predict
Summary:	Predict is a method offered by a trained model which allows the user to pass in a set of data points and returns a predicted class label for each.
Basic Flow:	1. User finds and supplies data point(s) for which they wish to predict a class label. Data should be given in vector or matrix format, depending on how many predictions are desired. 2. The user is given an option for which classifier (SVM, MLP) they wish to use, as well as any additional parameters associated with the models (verbosity, time required to predict, etc.). 3. The user selects a classifier 4. The user enters associated parameters. 5. The classifier completes the prediction and returns the predicted class label(s).
Alternative Flows:	Step 4: If the user passes incompatible data to classify go back to step 1, identify new data point(s).
Extension Points:	Evaluate (if training/testing data), None (if unlabeled data)
Preconditions:	The desired ML models have been created and trained using an appropriate dataset. Data given in vector or matrix format, consistent with data used to train model.
Postconditions:	The user receives a prediction corresponding to the supplied data point(s).

Compare - Completed

Use Case Name:	Compare
Summary:	Compare is a method of comparing the performance on the respective IBM and Google SDK's and libraries, and then selecting the performant of the two. Additionally, quantum and classical models will be compared. If the quantum piece does not show improvement over traditional techniques, then traditionally trained models will be used for inference on the emulator.
Basic Flow:	1. The same model is trained in both SDK/libraries 2. The more performant SDK/library is selected for the quantum simulator. 3. Quantum and classical models are then compared. 4. The more performant model will be used for inference on the emulator.
Alternative Flows:	If the quantum model fails to show improvement then the classically trained model will be used for inference on the emulator.
Extension Points:	Evaluate (model performance), Choose (choosing the more consistent model if there is no measurable performance gain between the two).
Preconditions:	The desired ML models have been created and trained using an appropriate dataset. The ML models would be the independent variable and the performance between the SDK/quantum comparison would be the dependent variable
Postconditions:	The quantum model shows performance improvement and is used for inference on the emulator. If not, the classic model is used instead.