Team 25

Team Members: Connor Sullivan, Amith Panuganti, Griffin Keeter, Derrick Quinn, Junyi Zhao

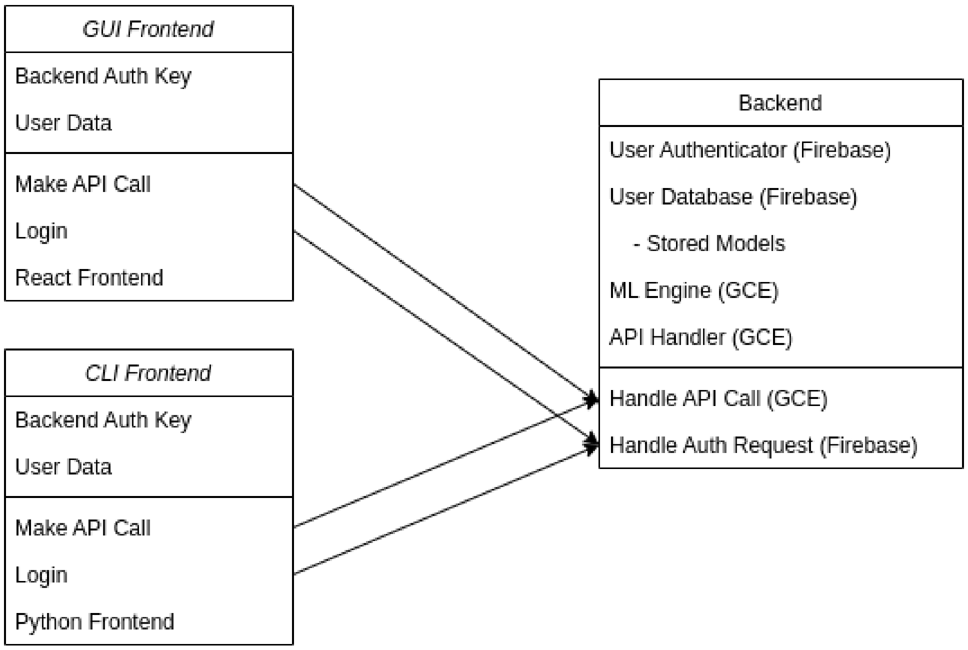
Project Name: ML For Everyone

Project Synopsis:

A web-based application that abstracts the technical details of machine learning away from the user to allow anyone to leverage the power of ML.

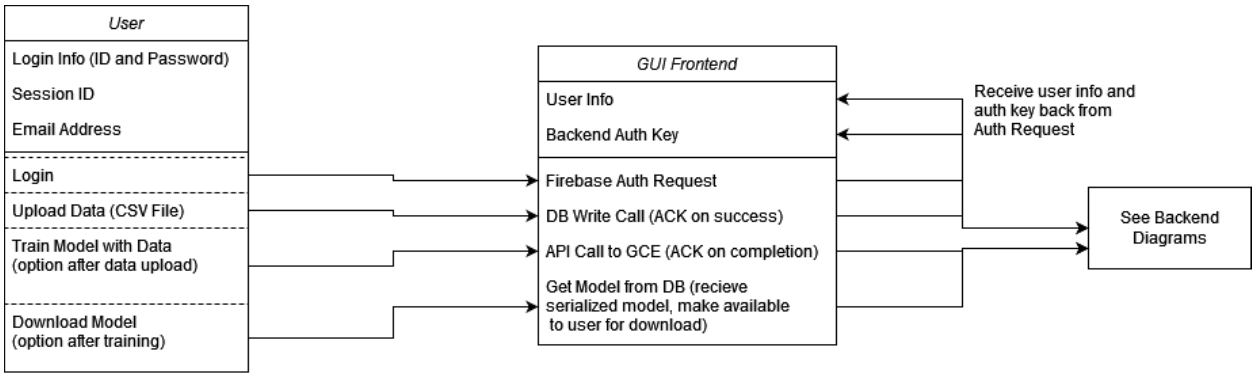
Architecture:

Overview:



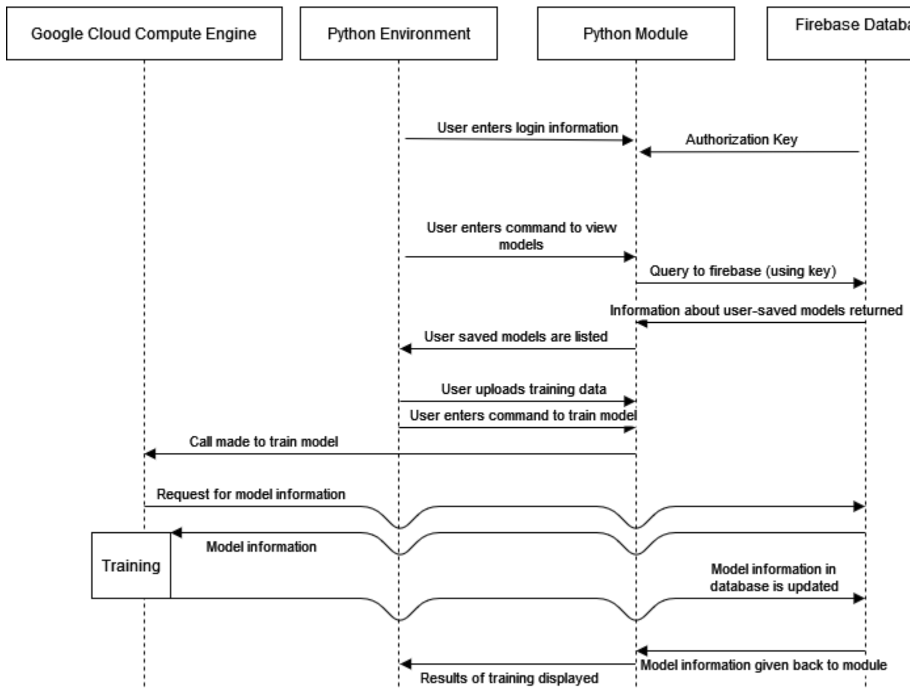
The overall architecture is straightforward: we will construct a backend, unifying Google Compute Engine and Firebase with our two front-end implementations: one with a graphical user interface in React.JS, and one as a python-based API. Both frontends will hold authentication information, as well as implement a way to make calls to the backend for login or Compute Engine tasks. Additionally, they will hold basic user data for easy access to that information. The unified backend will use GCE to implement an API handler, as well as build the models, and use Firebase to handle authentication and our user + model database. There will be two ways to interact with the backend: making authentication requests to Firebase and making Compute Engine API calls to GCE for everything else (training models, retrieving models, evaluating models on given points) and the implementation of all of this will be handled within the backend.

Gui Frontend:


User
Login Info (ID and Password)
Session ID
Email Address
Login
Upload Data (CSV File)
Train Model with Data (option after data upload)
Download Model (option after training)
GUI Frontend
User Info
Backend Auth Key
Firebase Auth Request
DB Write Call (ACK on success)
API Call to GCE (ACK on completion)
Get Model from DB (recieve serialized model, make available to user for download)
Receive user info and auth key back from Auth Request
See Backend Diagrams

The GUI frontend will be a simple, multi-tab page that places functionality first and will allow even the most inexperienced users to upload data and get a trained model back from the backend portion of the application. This component will be the simplest way to interact with the backend portion of the application, whereas the CLI frontend will be better suited for power users. It will be built using React as the visual frontend and will interact with both the API (for GCE needs) and with Firebase (for authentication and storage needs). Users will be able to login through the frontend, which will then make an authorization request API call to the Firebase backend. Once the frontend has received the data back from Firebase, it will store that data for the length of the user's session. Through the frontend, the user will be able to upload data, which can be stored in Firebase, and use the uploaded data to train models, which will be done through API calls to GCE. Once the trained model is handed off to Firebase from GCE, the user can interact with the frontend to make a call to Firebase to download the trained model for their personal use. Users can use the frontend to make calls to the database to store both their uploaded data and the trained model that was given to them after GCE processed the data. This portion of the frontend will also provide sections for the user to easily view metrics about their model.

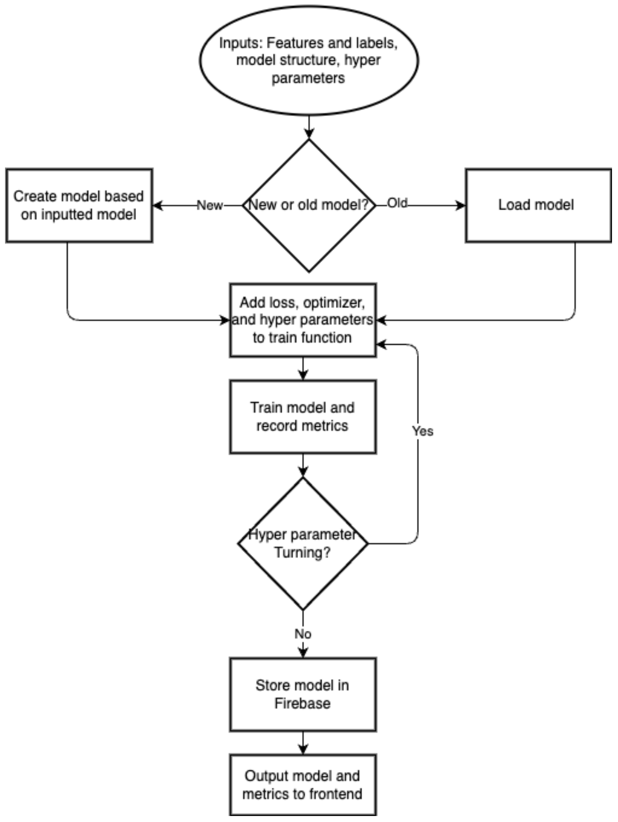
CLI Frontend:


Google Cloud Compute Engine
Python Environment
Python Module
Firebase Database
User enters login information
Authorization Key
User enters command to view models
Query to firebase (using key)
Information about user-saved models returned
User saved models are listed
User uploads training data
User enters command to train model
Call made to train model
Request for model information
Model information
Training
Model information in database is updated
Model information given back to module
Results of training displayed

The Python command line interface module will allow users to interact with models they have created, or to create new models. We will provide an API for making calls to the database and the Cloud Compute Engine. We will make this part of the frontend available as a python module which can be downloaded from the webpage, so users can import the module in a python environment and then call the functions defined in it on a command line or whatever other environment they choose. The module will be downloadable as python files. Once a user enters their user information, they will be able to access the database which will hold information like user settings and saved models. Like the webpage frontend, it will store an authorization key from Firebase. Firebase will provide the key when the user first logs in, then whenever the user makes a query to the database the key will be used to authorize the transaction. They will be able to enter commands using our team's API to run, train or design machine learning models like they would be able to on the webpage. There will be functions which allow users to create individual layers in a machine learning model, choose activation functions and optimizing functions. The user should have control over how to organize the training data they provide and how it is used to test the model, as well as how the model is tested. The API will support different filetypes containing the training data, like .csv files or directories. The user would be able to view the output of models and view what models they have built. The user would also be able to upload training data to the Cloud Compute Engine to test their models and download models once they are trained.

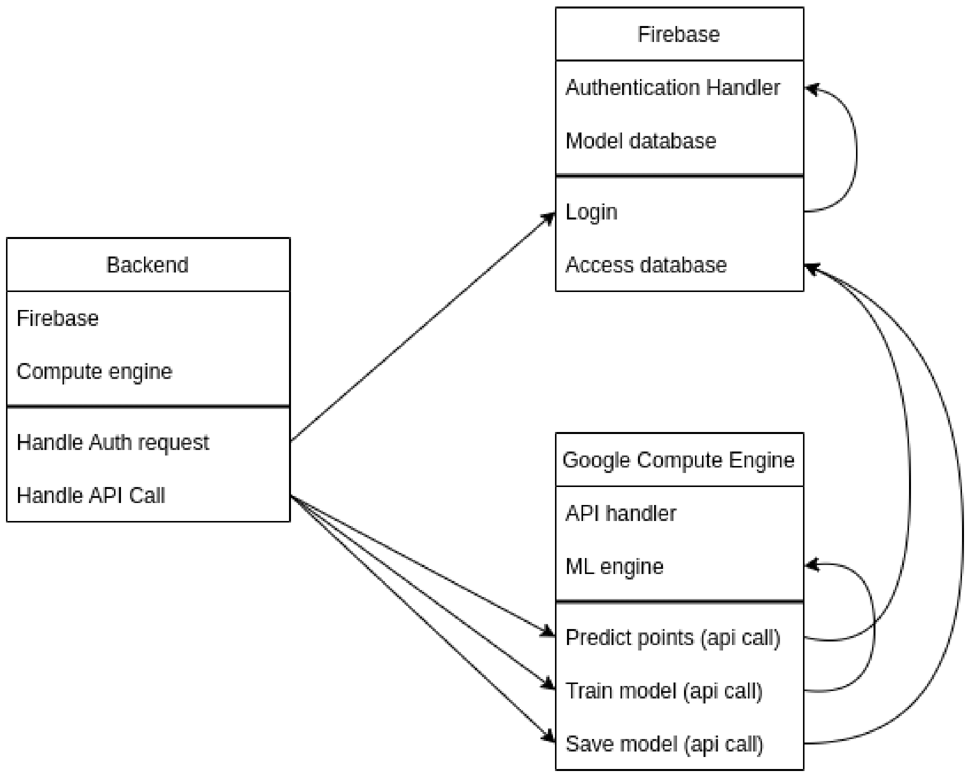
Google Cloud Compute Engine:

The software uses Google cloud Compute Engine to train machine learning models. The Google Cloud Compute Engine can train models faster by using multiple GPUS and CPUs on a remote server. The Google Cloud Compute will utilize a Python file or Jupiter Notebook to handle both the construction and training of the model. The following activity diagram represents the flow of the component, from input to output:


Inputs: Features and labels, model structure, hyper parameters
New or old model?
New
Old
Create model based on inputted model
Load model
Add loss, optimizer, and hyper parameters to train function
Train model and record metrics
Hyper parameter Turning?
Yes
No
Store model in Firebase
Output model and metrics to frontend

In the beginning, this component will receive the following inputs: the features and labels, the loss and optimizer functions, hyperparameters, and the model itself. Next, the program depends on how to load the model into the program. If the model already exists, the component will load the model from Firebase. If the model doesn't exist, then the program constructs a new model based on the user input model. How the model is constructed will depend on the machine learning library, its layers, and how it is organized. Once the model is constructed or loaded, we then set the loss, optimizer, and hyperparameters for the training function. Next, the component will train the model using the CPUs and GPUs over multiple epochs. Metrics about the model, including its loss and accuracy, will be recorded for each epoch. Afterward, the program can tune the hyperparameters and retrain the model to improve the model. Finally, the model will be stored in Firebase, and it and the metrics will be presented to the user in the front end.

Unified Backend:



The unified backend will contain Firebase for handling authentication as well as the database of models, and Google Compute Engine which will handle all processing of model-related calls. The calls themselves will be handled by an API call handler within Google Compute Engine and all machine learning will be handled by an ML engine running within the Compute Engine as well. The only calls handled directly by firebase will be for authentication, everything else will be handled via standardized API calls to the Compute Engine in order to simplify the backend.