**Description of Methods**

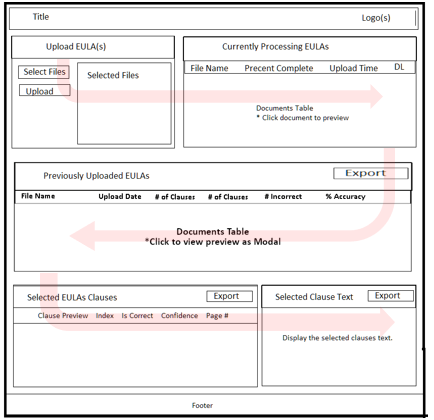
  A comprehensive description of the data, methods and software used to complete the solution.

**User Interface Development**

We developed EUL-AI with a web interface focused around batch processing. EUL-AI can be found at <https://anika-systems-eulai.ngrok.io/>. The website contains a tutorial when u first visit it which will describe each component in depth.

**Front-End**

The front end has been developed to give the business user a streamlined process. They can complete the entire process in a couple clicks. We developed the front end using a bootstrap theme that we customized.

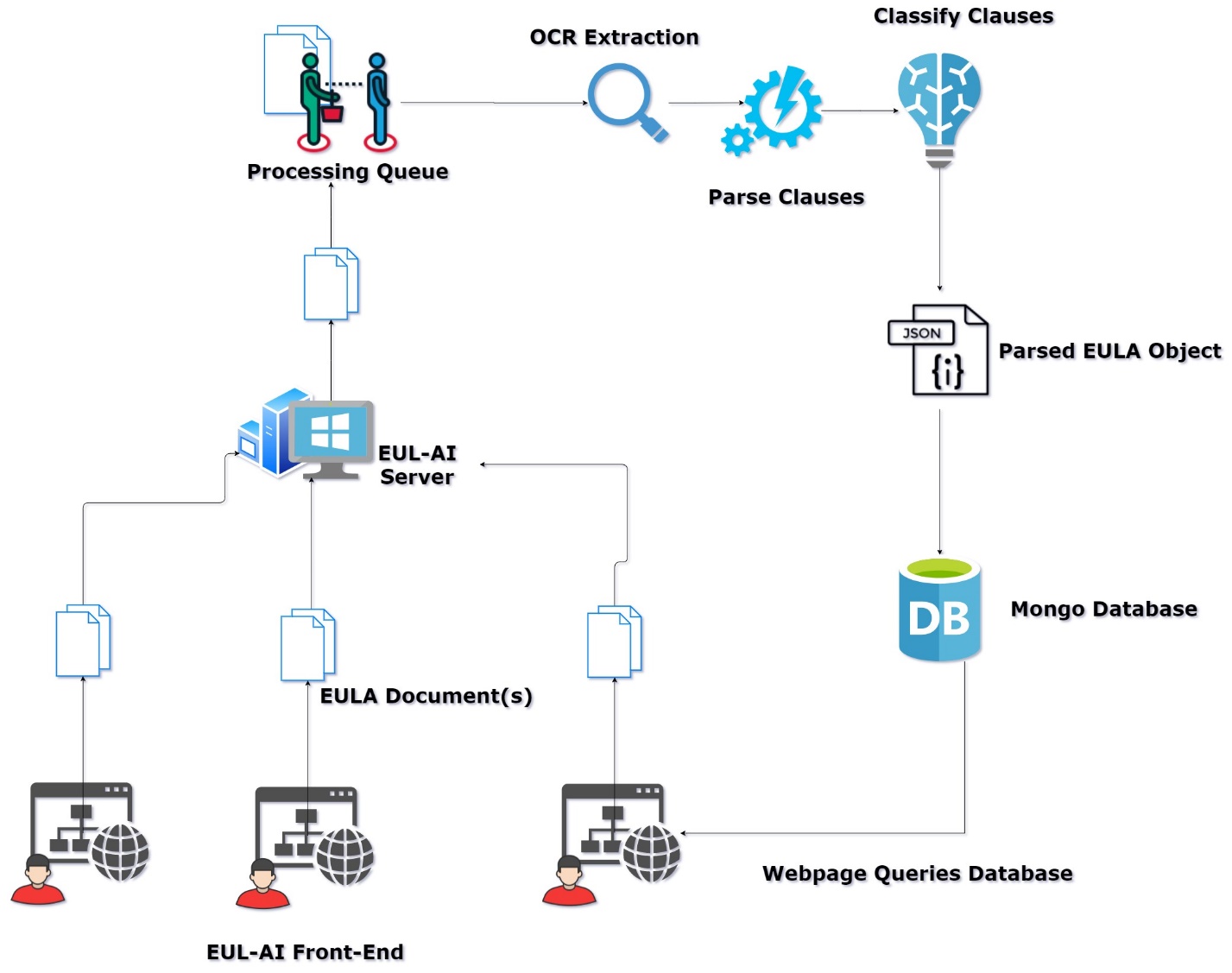
****

The user is able to upload multiple files at once. After they are uploaded, the user can see the, being uploaded in the queue. After the document has been processed in the queue, if there are no errors, it will be inserted into the database. The site provides a view into this database which is searchable as well. At the bottom of the UI is where the user can inspect all the clauses in detail and view their acceptance predictions. The user interface can be further customized using our theme.

In the future we may want to add login capabilities as well as more UI features using our custom theme. We would like the user to be able to edit the predictions of the clauses, as well as confirm their confidence. Also, the user should be able to edit filenames, and add titles to each EULA.

**Back-End**

EUL-AI website is hosted on Azure public cloud on a Windows 2019 virtual machine. Within the VM, we are running EUL-AI front end which is a Python Flask application running on a WSGI server. The application then connects to a MongoDB instance, which it uses for a processing queue as well as to store other EULA documents.



Once the documents are stored in the processing queue, a separate queue monitor program will process each document in the queue, one by one, extracting, disassembling, and classifying the clauses. They are then stored within MonogDB. The user can then view the database objects within their browser.

The site is behind a s secure grok tunnel, which allows for simple SSL encryption to site.

**Model Development**

**Data**

The only data we implemented in our model was the training data provided for us at the beginning of the challenge. While downloading the csv file, part of the data was unformatted and unusable. Therefore, due to the limited amount of data we had to work with, we had to make sure the data was formatted in the most efficient way possible. The first update we made to the data was to remove any control character that could have caused formatting issues. While this slightly improved our results, we found that the BERT pretrained model we were using was trained over lower-cased English text, therefore we created a new copy on Excel to convert all the text to lower-case. The final update we made to our training data was going through the larger sets of clauses and breaking them down to increase training data size and have them fit within the max sequence length variable we created.

**Methods**

Our machine learning model uses the BERT model to train deep bidirectional relations between words in unlabeled text. This form of natural language processing allowed us to use the training data provided to create a functioning model using only the valid or invalid labels. The model can create its own context using words around it due to its advanced tokenizer. By implementing transfer learning on this pretrained model, it required much less training data to produce more effective results. With the results of the model, customers can upload their EULA, and have it broken down by individual clauses to determine whether the clause is seen as acceptable or unacceptable along with the confidence score of the classification.

Our code breakdown goes as follows: we begin by importing all the Python libraries needed for our project and loading the dataset needed to be trained. We had the training data reformatted to include solely the Classification and Clause Text. From there, we split the data into training, validation, and test data to get results for accuracy on the model. At this point, we imported the BERT Model and tokenizer, specifically the “bert-base-uncased" model which is a 12-layer, 768-hidden, 12-head, and 110M-parameter model trained on lowercase English text. With the tokenizer, we found the length of all the clauses within the training data, and tokenized and encoded based on a max length of a sequence that we set. Once we got integer sequences for all the three sets of data, we converted them into individual sets of tensors. With the tensors, we created data loaders that wrapped the tensors and in preparation for running the model. We froze all other layers to make sure only the classification layer is being trained. With the data prepped, we defined the model architecture to have three layers: a dropout layer and two dense layers. We passed the layers into the forward function along with the tensors that had been previously loaded. We then passed the BERT model into our model architecture and which is trained on a local GPU. Likewise, we also imported the transformer AdamW to help optimize the learning rate. To take care of the class imbalance, we computed the class weights for the train set’s labels and processed the weights to the loss function. To measure the results of our training, we instantiated a loss function to run over 100 epochs. While fine tuning our model, we simply broke down each epoch to evaluate the training data by batches. During each epoch we also calculated the loss for both the training and validation data sets. From this we were able to make predictions of what our test data results would come out to be.

**Software**

While creating the Development Plan for this project we had a list of different applications we wanted to implement into our model. The first was Google Colab as a given to run our model as it provided the necessary GPU to run and train our model. In addition, we used MS Excel to work with the csv of training data and update any formatting directly on the application. In addition, we used PyCharm as an IDE for Python to create a script to remove control characters from the training data. Some of the Python libraries we used were Pandas, NumPy, Torch, train\_test\_split, classification\_report, AutoModel, and BertTokenizerFast. The data was loaded through a shared Google Drive folder, and most of the adjustments to the model were done online on Colab. Finally, the trained model was saved onto a PyTorch file, which could be exported and implemented in the UI tool for effective usage.

**Brier score** = 0.3069139966273187

**F1 score** = 0.66

The Brier score and F1 Score are essential measurements for this project as they determine both the model’s accuracy for probabilistic predictions and the model’s accuracy for the training data provided. loss. The Brier score specifically indicates the accuracy of probabilistic functions and the F1 score calculates accuracy utilizing precision and recall rates of the model. Due to the limitations of the amount of training data we were provided, it made it more difficult to optimize these scores. Similarly, we noticed that the training data was overwhelmingly filled with examples of unacceptable clauses compared to acceptable clauses, which really limited the model’s performance. To combat this, we used transfer learning by utilizing a pretrained BERT model and

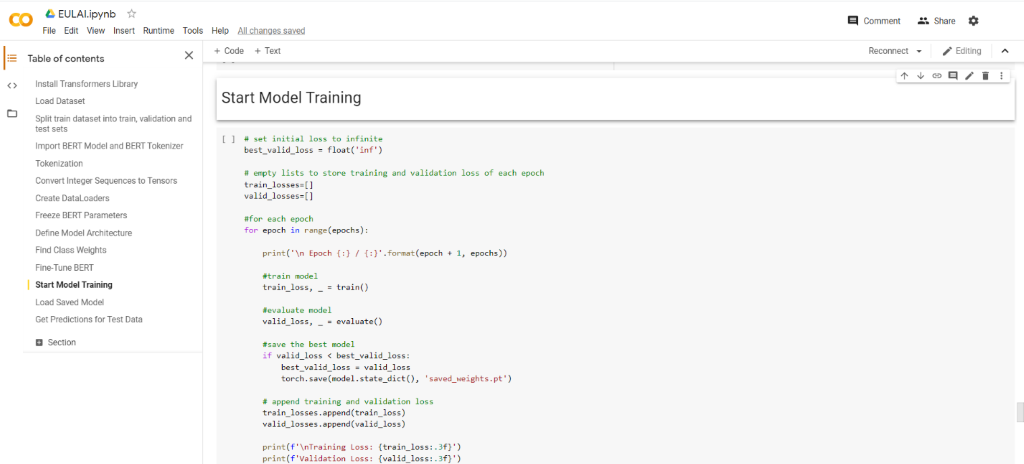


Figure 1 Displays the work environment of Google Colab and the section where the Model Training began. It shows the code used to provide a detailed breakdown of the information displayed in each epoch

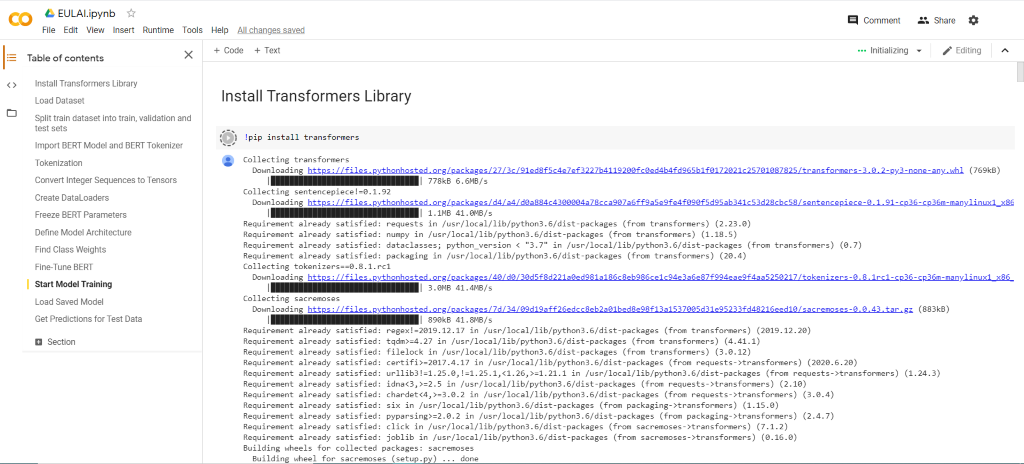


Figure 2 Shows the beginning of our online shared environment with all the libraries and transformers implemented in this project.

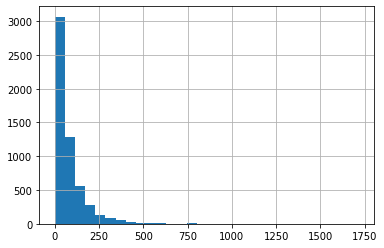
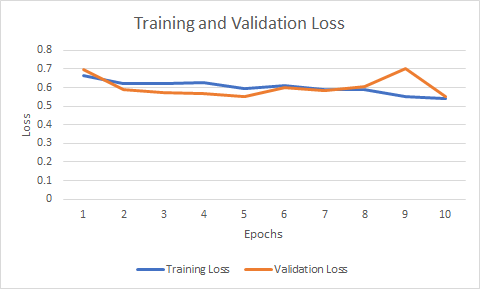


Figure 4 Shows the breakdown of the length of individual clauses and the number of clauses within the dataset

Histogram of Length of Clause Sequences

Figure 3 Shows the training and validation loss over the course of training the model.

Number of Clauses

Length of sequence