Cloud Infrastructure for Portable AI

Model: https://github.com/orugantiv/CSCE-585
React App: https://github.com/nathandolbir/machbot-585
Video Link: https://youtu.be/itqHZxjtUZQ

Nathan Dolbir
ndolbir@email.sc.edu

Kennedy Fairey
kfairey@email.sc.edu

Anirudh Oruganti
oruganti@email.sc.edu

Abstract

Machine learning (ML) systems require high-powered hardware to operate at a high speed for all users. Businesses and developers would require ample server space across multiple processing units, such as GPUs or TPUs, to serve customers effectively. In order to leverage large-scale ML models into the fingertips of all users with little concern over limitations, a serverless infrastructure must be implemented. We present a top-to-bottom cloud infrastructure that can run multiple state of the art (SOTA) sentiment classifier models on a remote GPU, and returns the response through a front-end webapp. This infrastructure demonstration shows how large-scale ML systems can be produced by development teams of any size, and can be deployed on portable, low-powered devices by offloading processing to the cloud.

1. Introduction

Large-scale neural network (NN) models require an extreme amount of computational power to yield fast and precise results. Portable devices such as smartphones or smartwatches, would be excellent application areas of large NNs, but they can't run strictly on their own hardware. The solution lies in cloud computing. Highly developed machine learning systems such as Apple's Siri and Amazon's Alexa are able to deliver intelligent responses by offloading computing power to the cloud, but these systems have an unbelievable amount of people, resources, and finances pumped into having these systems work without flaw. In order to spring forward innovation in all areas of ML, cloud infrastructure must be made possible for many developers with different interests, not just the tech giants. We have created a full-stack cloud solution to deploying large-scale NNs on portable devices, with the intention that it will be understandable and reproducible.

Our infrastructure's implementation area is sentiment classification. We have created five models that classify user input into six distinct emotions using probability, and they output the most likely emotion. These models are based upon the transformer architecture [2]. We also have created one natural language generator using Google T5 [9]. The flexibility of the cloud infrastructure will be evaluated using these six transformer models. We will prove that most any ML model can be deployed on portable, low-powered devices through the power of cloud computing.

The infrastructure is implemented through a chatbot webapp which takes user input and produces a response using the T5 model, along with highest likely emotion classifications for each selected model and their probabilities 0-1.

2. Background and Related Work

2.1. Cloud Systems

Cloud computing began to be sold as a service back in 1999 by the SalesForce Company[5]. They were popularized by Amazon's AWS in 2002, and then expanded upon by several other large companies such as Google, Microsoft, and HP in 2009.

Cloud computing gives the resources of one of the tech giants to everyone. It allows small companies to have as many bytes of data they desire, without having to maintain the physical hardware. It also allows anyone access to high-powered processing and cybersecurity.

2.2 Transformer Architecture

First introduced in the paper *Attention is All You Need* in 2017 [2], the Transformer model became the topic of conversation in the world of ML. It vastly improved the performance precedent previously set by Recurrent Neural Networks. Transformers omit the need to use recursion when reading input. This is possible through parallelization which speeds up the process of training[1]. When reading text, RNNs take in one word at a time then creates a word embedding for each word before going back and taking in the next word. In a transformer, we are able to pass in every word in a sentence at the same time and simultaneously create word embeddings for each word in that sentence. There are several other features of this type of architecture that make Transformer models superior. Transformers use encoder and decoder blocks to turn input words and training output respectively into vectors. Each block creates an input embedding, then creates a positional embedding based on the surrounding words. This gives the word vector context to combat words that have multiple meanings or intentions.

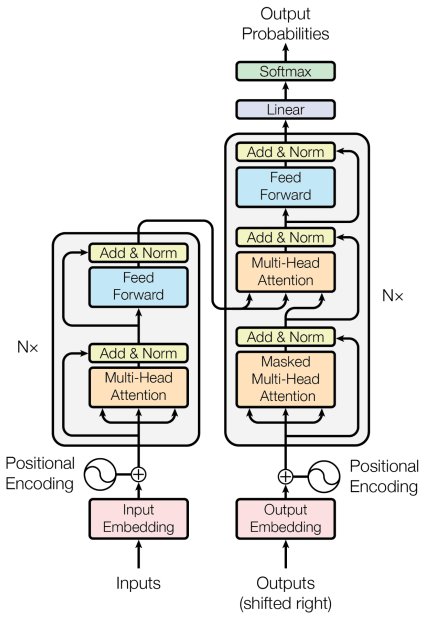


Figure 1. The Transformer architecture featuring an Encoder (left) and a Decoder (right).[1]

3. Data

Our dataset comes from kaggle.com from a data scientist named Praveen Govi[3]. It is 2 MB of text data that is labeled into six different emotions. These emotions are Anger, Fear, Joy, Love, Sadness, and Surprise. There are roughly 16,000

sentences of training data, 2,000 sentences of test data, and 2,000 sentences of validation data.

This dataset was chosen for a few reasons. The first reason is that we felt the six emotional categories were very precise in nature and could encapsulate most text. Another is that the data was easy to manipulate into a more desirable format to feed our model. We were also able to easily download and upload the entire dataset to our cloud model with ease. Finally, as seen in Figure 2 there seems to be a normal distribution giving us an opportunity to have a padding length closer to max sentence length.

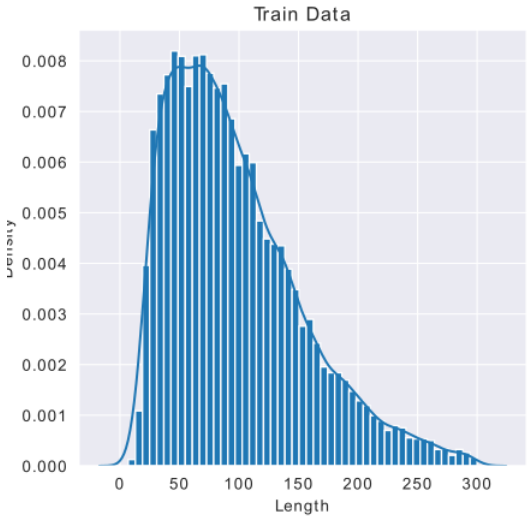


Figure 2. Above is a bar graph of the training data with normal distribution of density vs. length.

4. Methods

4.1 Infrastructure

4.1.1. Google Cloud Platform

Google Cloud Platform (GCP) is the cloud computing service that serves as the top of our infrastructure on the backend. GCP hosts a number of cloud computing services that we utilize to host our sentiment classifiers.

Google Storage was used to store trained models into spaces called buckets. We considered storing our data in these same buckets, but found it was easier and faster to upload the data to Google Drive, and then mount the Drive to the worked-on Colab notebook.

The other service used was Google Cloud Function. Functions react to specified triggers and execute code once initiated.

4.1.2. Google Colab

Google Colab is a Jupyter Notebook environment provided by Google. It was easiest to use this as the environment for training because it is already connected to all Google cloud computing apps through Google accounts. We were able to easily save the model to our GCP storage bucket and connect our data from Google Drive using Colab as an intermediary.

4.1.3. Firebase

Firebase acted as the middleman between our front-end and back-end in this cloud computing infrastructure. The Firestore Database on Firebase is Google's cloud hosted NoSQL document database. Firebase apps can be connected to frontend web or mobile apps, so data can be passed between the two. When data is input to a frontend, it is saved in the Firestore Database as a document that is formatted similar to a .json file. Each document has a unique UID and is part of a collection of documents, of which there can be multiple.

These documents (or fields in the documents) can also be sent up to a GCP

project so that the data can be manipulated. Once the data is manipulated, it can then be passed back down to Firestore Database and then passed down to the frontend.

4.1.4. React App

The bottom of our cloud computing infrastructure is the React frontend app. React is a Javascript framework geared towards seamless frontend web development. The webapp imports the Firestore database as an object so that proper data fields can be created. Once there is data in those fields to be passed back, then the React app can pass data back to Firebase.

4.2 Implementation

Starting from the bottom (the frontend), the app makes the user sign in and use a username before they can interact. They then can type in any sentence and the app will take the user's input, create a Firestore document, and pass that document to Firebase as part of that user's unique document (in this use case, a message) collection. The user can also choose from five unique classifiers to select for emotion classification.

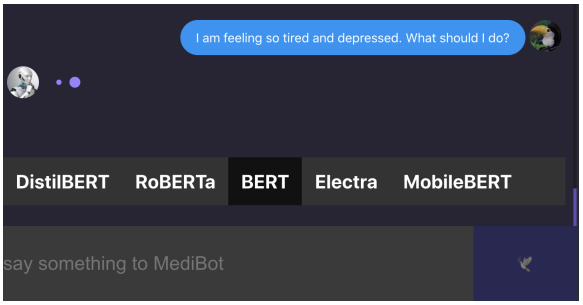


Figure 3. A user asks a question and selects BERT to classify the text. The bot displays a loading animation while the response is being generated.

It also passes the data to a function buffer document along with the user's collection and document ID so that the model responses can be restored in the correct document.

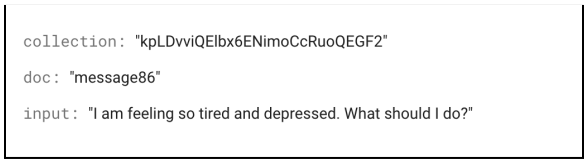


Figure 4. The message content is saved with metadata so the response is rerouted back to the correct document.

The document is now inside the Firestore database and can be recognized by the next level of the infrastructure. The cloud function in the GCP project waits for an update to the function buffer document. Once it is triggered, the python script in the cloud function runs.

The cloud function initializes an instance of the firebase app within the scope of GCP. This makes it possible for multiple function instances to create model responses if multiple questions are being asked at the same time (by different users, or just multiple from one user). The function then loads the saved model that is stored in the Google storage bucket. The input taken from the Firebase document is passed into the trained model to generate prediction weights for six different emotions. There is then a method that takes the highest probability from the six emotions and returns it. This probability and the associated emotion are then passed back to the corresponding message document.

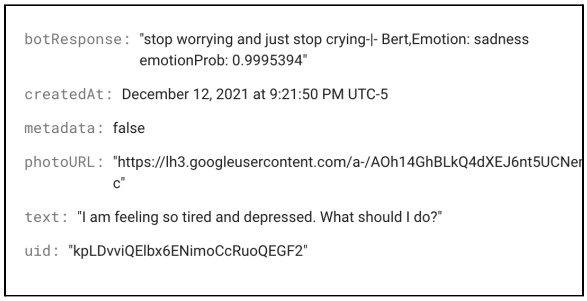


Figure 5. The message's document.

The React app displays user messages and the bot's response, and will replace the loading animation with the response immediately as it appears in the message's document.

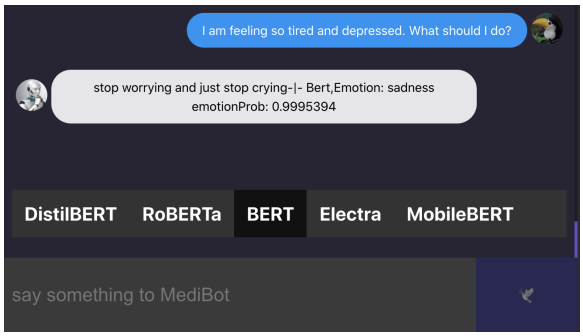


Figure 6. The bot responds to the user with a T5-generated reply and a classification with probability for the selected BERT model.

Google Colab was used for all development and training aspects of the actual Transformer model. The data pipeline does not involve Colab, but the model was trained, tested, and saved to GCP using the services of Colab.

5. Experiments

5.1 Models

To test the robustness of our infrastructure for different ML models, we fine tuned five different types of Transformer models, similar to the kinds discussed in Cortiz's 2021 paper[4]. Giving us an opportunity to achieve high accuracy and f1-macro as we desired due to the pre-training of these models.

In Cortiz's 2021 paper[4], they used a large dataset with many text classifiers unlike us. They used five different transformer pre-trained models offered by huggingface.co to train and test their dataset. Using that entire dataset all at once to train without sub-grouping very similar classifiers to train separately in different layers or in similar fashion, which would avoid model recognition of certain patterns making the model more robust. They finally ended with not as high accuracies and f1-scores.

We decided to fine tune five transformer models with a smaller dataset and more distinct text classifiers compared to Cortiz's 2021 paper[4]. We fine-tuned four same and one different transformer model. The models trained and tested were: DistilBERT[11], RoBERTa[12], BERT[10], ELECTRA[6], and MobileBERT[13]. We followed many techniques shown in a tutorial[16] since we lacked deep knowledge in ML.

5.2 Metrics

There were four metrics used to measure the effectiveness of each model that was trained. These metrics are: accuracy, F1 macro score,Time to complete Training and Evaluation, and different thresholds on probability from predictions.

5.3 Parameters Settings

Each model was fed the same data, given the model specific tokenizer, and trained with the same batch size, number of epochs and learning rate like in Cortiz's 2021 paper[4]. This was accomplished with a single "training" function that took in the model name and model type as parameters.

- **Loss: Sparse Categorical Cross Entropy**
- **Optimizer: Adam**
- **Learning rate: 5e-5**
- **Batch Size: 16**
- **Epochs: 4**

6. Results and Discussion

6.1. Cloud Response Time

Due to the fact that our project is not hosted on a constant server (it would cost more money than we are willing to pay for this project), each time that the cloud function is triggered, there is a new instance of the model and each of the necessary packages must be redownloaded and installed. This causes the response time to be around an uninspiring 45-60 seconds, depending on models selected and complexity of input. We found it unnecessary to record these times as a metric of success because the actual response time from the model is much less than the time it takes to register a response on the frontend app.

6.2. Model Results

6.2.1 Model Training Results

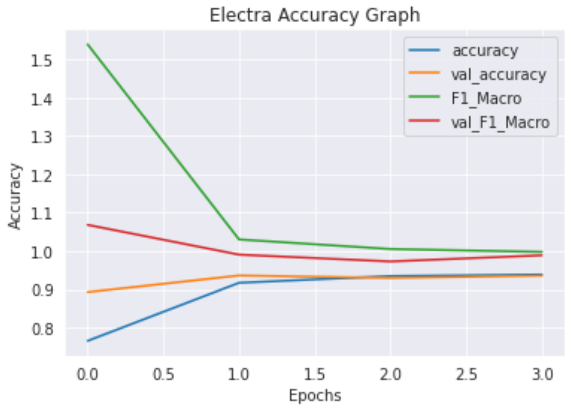
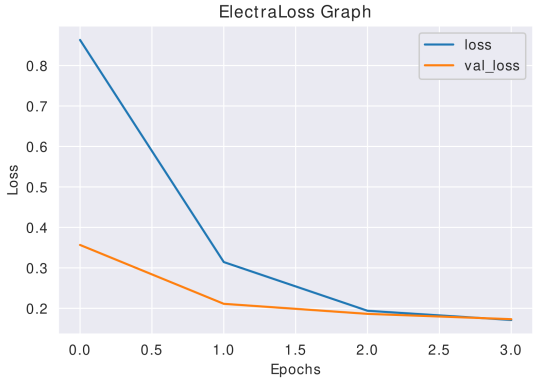


Figure 7. Above is an example of the visualized data from the ELECTRA model post-training. These Accuracy graphs were generated for each Transformer model



.**Figure 8.** Above is an example of the visualized data from the ELECTRA model post-training. These Loss graphs were generated for each Transformer model.

While training the models, all transformer models we used showed the same or similar nature and had the same form as seen in Figure 3 and Figure_4. Out of all our models, ELECTRA definitely performed better having linearly increasing Val_Accuracy and linearly decreasing Val_loss. The F1_Macro and val_F1_Macro are custom metrics we found[14], they did not really correlate to F1_Score from sklearner python package but we observe a correlation with loss. Often during, sometimes loss increased with no effect on accuracy or validation which was very interesting since tensorflow framework is based on lower loss and accuracy is linearly dependent[15].

	Accuracy	Val Accuracy	Loss	Val Loss
DistillBERT	0.9265	0.9340	0.2069	0.1952
RoBERTa	0.9048	0.9300	0.2680	0.1903
BERT	0.9151	0.9364	0.2399	0.1547
ELECTRA	0.8887	0.9234	0.3096	0.2286
MobileBERT	0.8410	0.8731	0.4623	0.3513

Figure 9. Above are the metrics used to measure the performance of each of the five Transformer models. “Val” stands for validation and each value is the average over four epochs.

Accuracy and loss are two of the most standard metrics used to evaluate the performance of ML models. Accuracy is a measure of the number of correct predictions over the total number of predictions made[7]. Validation accuracy helps to verify the accuracy of the training data by creating a separate set by which to compare the model to. If our only metrics came from the predictions made to the training set, then our metrics may be misconstrued.

Loss is the difference between the predicted value by the model and the true value[7]. There are many types of loss functions used in ML metrics but the one used for these models was cross-entropy.

As shown by the data in Figure 5, the first three models (DistillBERT, RoBERTa, and BERT) perform better then the other two. This is proven across all four metrics; however, each model performed relatively well.

6.2.2 Accuracy and F1 Scores Results

Emotion	**BERT**	**DistilBERT**	**RoBERTa**	**a XLNet**	**Electra**
macro avg	**0.46**	**0.48**	**0.49**	**0.48**	**0.33**

Figure 10. Above are the f1-macro avg from [4] D. Cortiz results from fine tuning GoEmotion data.

Emotion	Bert	DistilBert	Electra	Roberta	MobileBert
anger	0.894632	0.927007	0.925714	0.911972	0.906526
fear	0.887417	0.868778	0.887931	0.871671	0.908686
joy	0.944649	0.944803	0.943369	0.947589	0.950181
love	0.824798	0.795987	0.812698	0.804428	0.849398
sadness	0.962901	0.961571	0.962712	0.964041	0.963668
surprise	0.727273	0.744828	0.743802	0.751678	0.702703
accuracy	0.919	0.9205	0.924	0.9225	0.928
macro avg	0.873612	0.873829	0.879371	0.87523	0.880193
weighted avg	0.919962	0.920282	0.923378	0.921122	0.92727

Figure 11. Above are the f1-macro avg from [4] D. Cortiz results after fine tuning GoEmotion data.

Models	**Training(min)**	**Evalution(min)**	**Evalution (msg/s)**
Bert	**39.898933**	**3.108629**	**0.093259**
DistilBert	**20.787233**	**2.53336**	**0.076001**
Roberta	**40.637583**	**3.122908**	**0.08981**

Electra	**12.080952**	**2.993671**	**0.093687**
MobileBert	**30.145858**	**7.588735**	**0.227662**

Figure 12. Above are the elapsed time results for training and evaluation.

To understand what an F1 score is,we must first understand two other ML metrics. These metrics are precision and recall. The underneath equation describes precision,

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

and the next equation describes recall.

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Both of these equations come from Koo Ping Shun's article from 2018 titled *Accuracy, Precision, Recall, or F1?*[8]. We can understand what true positive, false positive, and false negative mean in the scope of predicting the response to an input.

Now that we have a brief background on precision and recall, we can introduce the formula for an F1 score.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

This metric seeks a balance between both precision and recall, because in real world applications of ML, situations with false positives and false negatives can prove to be much more costly.

F1 scores are considered better the closer they are to 1. As we can see from the Figure 11, All the models' macro average seems to be really good but when considering training and evaluating times Electra definitely seems to be better performing overall.

6.3. Implementation Challenges

The original plan was to create a conversational chatbot with this same infrastructure, but we ran into some challenges when attempting to implement it. Our first struggle was we failed to find a true purpose of the chatbot. Most chatbots are geared towards accomplishing specific tasks within the scope of a specific business, and since we had no business intentions with this product, it made no sense to create a chatbot with that goal. We also struggled to find the right data to implement a chatbot.

During the model build and deploy, we encountered trouble in loading a saved model using keras library; since our model includes a transformer custom model, it was having problems reading the model architecture. To resolve this issue, we saved the model's weights and hard-coded the same architecture before loading the weights. While trying to deploy a model from the google cloud bucket, we were not able to directly read files and folders which caused more latency in google functions since we needed to download from the buckets to google cloud functions.

There were many UI bugs to fix, and certain components were reworked

7. Conclusion

There are clearly areas in this project that can be improved upon if revisited, but overall we feel that our mission was achieved. The goal was to create an infrastructure that used cloud computing to host an ML model that would require to much processing power for a small portable device. From end to end, it was proved that the data pipeline is possible and it can be implemented with most any type of ML project.

It was also a goal of ours to create multiple versions of modern ML models

(Transformers) to prove the robustness of this type of infrastructure. We accomplished this by allowing each of the five models to be selected and run individually in the infrastructure.

8. Further Exploration

The infrastructure developed in this project is very flexible and could be used for an infinite number of interesting ML projects. Our main goal was to prove that portable AI could be achieved with a minimal budget and a small team of inexperienced developers. We could now build and train other types of models that we find interest in and easily train and deploy them with cloud computing.

If we were to develop a more concrete plan for production, we could also purchase some of Google's cloud services to improve the project as a whole. Examples of this would be to buy server space so that we could actually host one of the models and not have to create a new instance each time an input is received. We could also purchase a TPU on GCP to implement much larger models that train at a much faster pace.

This project opens the door for many future projects that we or our peers could take on that could serve both the research and industrial world.

References

[1] U. Ankit. *Transformer Neural Network: Step-By-Step Breakdown of the Beast.* 2020

[2] A. Vaswani, N. Shazeer, N. Parmar, J.Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, I. Polosukhin. *Attention Is All You Need. 2017.*

[3] P. Govi. 2019. *Emotions Dataset for NLP.* Kaggle. <https://www.kaggle.com/praveengovi>

[4] D. Cortiz. *Exploring Transformers in Emotion Recognition: a comparison of BERT, DistillBERT, RoBERTa, XLNet and ELECTRA.* 2021.

[5] P. Srivastava, R. Khan. *A Review Paper on Cloud Computing*. 2018.

[6] HuggingFace.co. *ELECTRA*.

[7] H. Kumar. *Loss Vs. Accuracy*. Github.com. 2018.

[8] K.P. Shung. *Accuracy, Precision, Recall, or F1?*. 2018.

[9] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. Liu. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2019.

[10] huggingface.co. *BERT*.

[11] huggingface.co. *DistillBERT*.

[12] huggingface.co. *RoBERTa*.

[13] huggingface.co. *MobileBERT*.

[14]https://datascience.stackexchange.com/questions/45165/how-to-get-accuracy-f1-precision-and-recall-for-a-keras-model.

[15] freeCodeCamp.org. June, 18 2020. *Keras with TensorFlow Course - Python Deep Learning and Neural Networks for Beginners Tutorial*. Youtube.

[16] James Briggs. November 19, 2020. *How-to Build a Transformer for Language Classification in TensorFlow*. Youtube.