Group 5 - Jamie Wu, Seok Hyun Kim, Fabiola Gallardo, Imran Omari, Gautham Sivakumar

Written Report

**Introduction**

When ASL interpreter Justina Miles performed at Rihanna’s SuperBowl performance earlier this year, her expressive interpretations were considered a prime example of making media more accessible to individuals that are deaf or hard-of-hearing. When selecting a project concept, we were similarly inspired by the importance of ASL in making everyday activities more accessible to all groups of people. Our goal was to create a model to detect and translate American Sign Language (ASL) fingerspelling into text predictions, which has significant implications for live captioning and media accessibility. Throughout the project and conversations with peers, we noticed the lack of undergraduate-level knowledge about *Transformers*, a model that is heavily used in automated speech recognition (ASR) and text generation models and features an encoding-decoding format. Aiming to make our project more accessible to peers, we modified our project scope to emphasize making Transformer-based predictive models more accessible to undergraduate students. We first conducted a literature review of Transformers, and used our learnings to successfully replicate an ASL-to-text predictive model. Then, we added extensive documentation on the model to serve as a resource for others to better understand how they can implement basic Transformers in an ASL-to-text setting. Finally, we conducted an analysis of multiple prize-winning Transformer-based ASL-to-text models to determine how ASL-to-text models can be further improved. By conducting a thorough literature review of Transformers and applying our learning to successfully implement and document a Transformer-based prediction model, our team was able to develop a technical understanding of Transformers while also gaining understanding of modern implementations of Transformers and sharing that knowledge with the class.

**Background**

Transformers are a modern form of deep learning architecture specialized for taking in an input data sequence and producing an output sequence. Specifically, they take a sequence of tokens and predict the next token in the output sequence by iterating through encoder layers. The encoder code generates encodings that define which parts of the input sequence are relevant to each other, which are then passed to the next encoder layer. Afterwards, the decoder takes all these encodings and uses their derived context to generate output sequences. The key factor that separates Transformers from other sequence-to-sequence models is that they do not necessarily process data in order; rather, they use an “attention mechanism” to provide better context around the items of input sequences and make better output predictions. Attention mechanisms are a set of mathematical techniques designed to detect subtle ways even distant data elements in a series influence and depend on each other. Since Transformers do not receive inputs directionally, they do not suffer from short term memory, and the models are able to use the context from the entire sequence to aid them in making predictions. Variants of attention mechanisms, such as self-attention and multi-headed attention, allow Transformers to associate individual input tokens with other input tokens or to run multiple sequences in parallel. We chose to use a Transformer model for our project due to the architecture’s superior ability to process contextual relations among sequential data.

To emulate an ASL-to-text predictive model, we sourced a dataset from a Kaggle competition hosted by Google. The competition task was to translate videos of 100+ deaf signers signing 68,000 unique phrases with ASL fingerspelling into text phrases. For the predictive model, we primarily worked with numerical coordinate data provided by Google that describes different landmarks on an individual’s hand. While this information was provided as project scaffolding, our team looked further into how this data was generated from the original ASL videos. The numerical coordinate data was generated through an image processing resource called Mediapipe, which processes videos of human actions and is able to identify key landmarks throughout the entire body. In the preprocessing phase, we isolated a subset of coordinates specifically related to the upper torso and arm portion of the body to capture any variations in finger positioning and upper body posing. Below is a diagram of the upper-body Mediapipe landmarks, with the ones we used circled in yellow highlight.

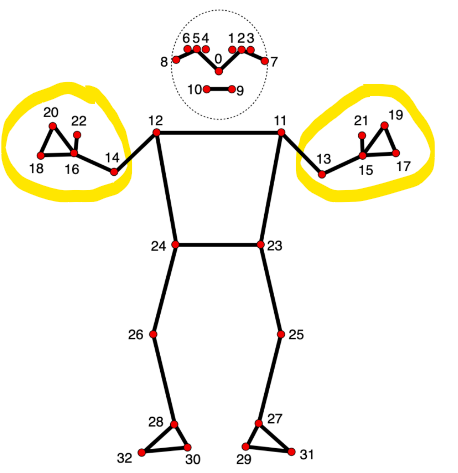


Figure 1

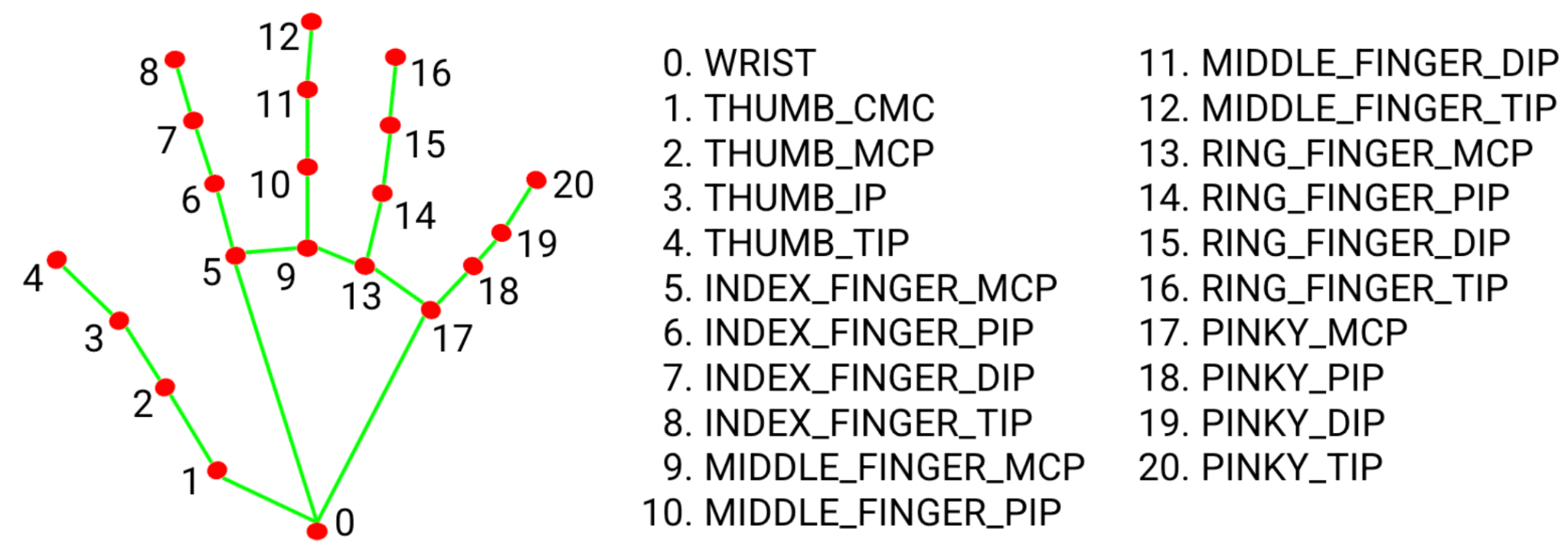


Figure 2

**Methodology**

Phase 1: General Transformer Literature Review

In order to know what we are implementing and how, each of us did external research on Transformers. We looked at research papers, Youtube videos and even the textbook to further our knowledge. From there we gained an understanding of how to implement a Transformer for this project specifically, which is emulating an ASL-to-Text Transformer-based predictive model.

Phase 2: Emulating an ASL-to-Text Transformer-based Predictive Model + Documentation

For efficient code collaboration, we created a GitHub repository to manage our updates and store dataset files like train.csv. We used Google Colab to write our Python code, which made it easy to run and test segments of the code individually and run code on a browser.

For the predictive model, the coordinate data generated by Mediapipe is output in binary .parquet format. While .csv files are more readable, .parquet files have columnar storage and built-in compression and encoding that makes it a better choice for such a large dataset - each of the parquet files contains more than 160,000 observations and 1,629 spatial coordinates. For the project we randomly selected 15 of the provided 68 parquet files to work with.

Each parquet file contains information for 1,000 phrases, so our full dataset contained 15,000 unique phrases. A separate file called train.csv contains the “Ground Truth” information mapping a text phrase to its associated parquet file with coordinate data. To build the model, we combined the extensive coordinate data from 15 parquet files to its associated “Ground Truth” information from train.csv into a format called TFRecord. TFRecords are a Tensorflow compatible method to store a sequence of binary records. To combine the binary parquet landmark data and the train.csv text phrase information into one dataframe, we looped through the phrases and turned it into binary format and then wrote it to a .tfrecord file. This consolidated TFRecord table acts as our new “primary dataframe” and we employ the dataframe to create our training and validation data. Lastly, we also used a .json file containing number-to-character mappings to translate between numerical Transformer outputs and text predictions. Once all the preprocessing and conversion is complete, we moved on to the Transformer code.

We trained a Transformer model to take in the input data and return predictions. The architecture used is similar to an Automatic Speech Recognition Transformer provided in publicly available Keras documentation. We ultimately ended up only needing to finetune a small part of the model as we were able to treat the ASL Fingerspelling recognition problem similar to speech recognition. In both cases, we have to predict a sentence from a sequence of data - just that our problem involved Mediapipe landmarks instead of sound files.

A key function of the Transformer encoder was processing landmark coordinate features as input; we applied convolutional layers to downsample them and process local relationships. When processing past tokens in the decoder, we sum positional embedding and token embedding to form the inputs for the Transformer. The Encoder function consists of a multi-headed attention operation followed by dropout, layer normalization, a feed forward layer, and a second dropout and layer normalization operation. These are accomplished using Keras neural network functions. The Transformer Decoder uses the input token encodings to predict a sequence of characters. Together, our model is trained using the training and validation datasets we set up earlier, utilizing a categorical cross-entropy loss function along with an Adam optimizer. As the program runs, it plots training and validation loss.

Phase 3: Literature Review of Award-Winning ASL-to-Text Transformer-based Models

To supplement our learning and understand what industry leading Transformer implementations look like, we looked up the top three winning notebooks from the Google Kaggle competition. One group member tackled each of the notebooks and took a deep dive into how they implemented the Transformer and reviewed their code. From there, it was explained to the rest of the group in order to have full understanding of what was happening.

**Results**

Phase 1: General Transformer Literature Review

From our individual research on Transformers, the team was more well versed with the encoder-decoder structure and how attention mechanisms work. A summary of what we learned is included in the “Background” section earlier in the report.

Phase 2: Emulating an ASL-to-Text Transformer-based Predictive Model + Documentation

Figure 3 shows the training and validation loss results for the model we implemented. We ran a total of 13 epochs, and we can see that around epoch 12 the validation loss starts to diverge from the training loss. This indicates that our model is overfitting.

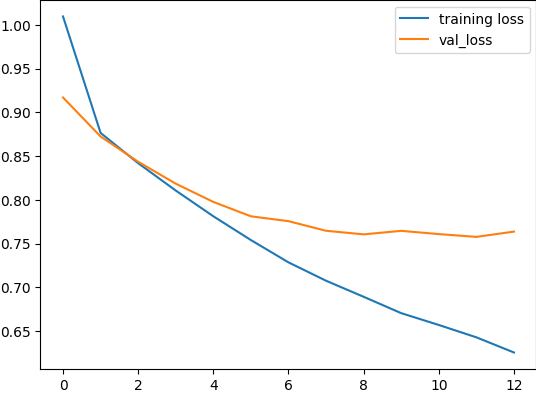


Figure 3



Figure 4



Figure 5

In the images above, we can see how the prediction model does not accurately predict most of the letters. There are some instances where the model accurately predicts the first half of the letters or with minor errors and then misses the entire second half of the sequence. (Figure 4) This could be due to mispredicting a certain letter in the ASL Fingerspelling, which throws off the rest of the sequence. There are also some instances where the prediction is completely incorrect. (Figure 5) The model seems to have learned the pattern of website names “www.” from other phrases, but did not accurately apply it in this situation.

Phase 3: Literature Review of Award-Winning ASL-to-Text Transformer-based Models

After initially constructing a functioning Transformer-based ASL-to-Text Model, we sought to improve the model and achieve better performance. In terms of pre- and post-processing, we noticed that multiple leading Kaggle teams augmented their input data beyond the provided Google dataset to optimize model performance. Augmentations included time augmentation, which randomly modifies the frame rate of the original video of the ASL signer to enhance the amount of training data that the model sees. Other augmentations included flipping coordinates from left hand to right hand.

In terms of the Transformer implementation, one key thing we noted was that the base Transformer model used was heavily based off of the Automatic Speech Recognition with Transformer tutorial provided on the Keras website. Thus, when seeking to improve our ASL-to-Text Model, we researched modern ASR methods for inspiration for our program. One of the top models used for speech recognition was the Conformer model, which combined the features of both architectures to efficiently model both local and global dependencies. The Squeezeformer further improved upon the Conformer structure and made things more efficient while maintaining the convolution-augmented attention mechanism. Several of the top solutions on Kaggle mentioned or implemented parts of the Squeezeformer architecture. For example, the top solution had an encoder based on the Squeezeformer architecture and a normal decoder composed of two transformer decoder layers. A common trend was that many of the top solutions recognized the correlation of speech recognition algorithms with the competition and incorporated ASR algorithms into their programs.

**Discussion**

After all three phases of our project - preliminary Transformer research, predictive model recreation, and reviewing award-winning Transformer implementations for ASL-to-Text. Our project has given the team valuable insights into collaborating on a coding project across various programming environments. For example, we initially chose Google Colab due to its seamless Python utilization that didn’t require any local setup or downloads. However, we later realized that Colab didn’t facilitate concurrent editing so we learned to use GitHub repositories to streamline our coding workflow. Additionally, team members had distinct preferences when it came to file loading – some preferred loading data locally for each runtime of the Colab VM, while others opted to link their Google Drive accounts. To accommodate these differences, we developed different modules, ensuring that files could be loaded in both scenarios, allowing everyone on the team to contribute effectively to the project.

Looking at the results of the ASL-to-Text Transformer model, we can see that our prediction accuracy was not that great, which makes sense because we trained with a more manageable (~25 GB) subset of the parquet files (100+ GB) and only ran 13 epochs. However, our research into award-winning models in Phase 3 helped us better understand how we could further optimize the data processing and Transformer model in the future. Regardless, through the process of recreating a Transformer-based predictive model and writing detailed documentation to make it a resource for others, we learned a lot about how ASL-to-Text Transformer models work and team collaboration.

**Conclusions**

Overall, through conducting a literature review of ASL-to-Text Transformer models and emulating one ourselves, we learned a lot about AI and Transformer. With this specific project we explored different types of Transformers like the Squeezeformer architecture, automated speech recognition, and Tensorflow as well. With more time, we would love to create a more unique rendition of the Transformer in our ASL-to-Text prediction model. While the prediction model was not as accurate as we would like, the process of emulating the model from Kaggle and documenting the implementation in great detail gave us a much better understanding of transformers and AI in general. The team thoroughly enjoyed this project, and feel optimistic about its implications for further ASL-to-text development. Having better automated ASL interpretation empowers more people to bridge the communication gap between sign language and spoken language, and we hope that through our research on Transformers and documentation of our predictive model, future ASL-to-Text development will become more accessible to others.

**Contributions:**

Phase 1: General Transformer Literature Review

Everybody contributed individually by researching Transformers.

Phase 2: Emulating an ASL-to-Text Transformer-based Predictive Model + Documentation

Jamie set up the GitHub and Colab notebook. Fabiola helped write code for data visualization with Mediapipe and documented the code. Jamie helped write code for preprocessing the data with TFRecords and character\_to\_prediction and documented the code. Harry modified the transformer code to make it compatible with 15 parquet files and helped write and run the Transformer code.

Phase 3: Literature Review of Award-Winning ASL-to-Text Transformer-based Models

Fabiola, Jamie, and Imran each reviewed one of the top 3 Award-Winning Kaggle implementations of the ASL-to-Text model. Gautham also conducted a lot of research into Transformers, Squeezeformers, and the ASR Transformer from the Keras documentation.

Report and Presentation

Jamie, Fabiola, and Gautham wrote the report. Imran and Harry worked on the presentation. Speaking for the presentation was distributed evenly so everyone had opportunities to talk.