Parrot: An Abstractive Summarization Application

Team Members: Vincent Marklynn (vmarklynn@seattleu.edu), Anjali Sebastian (asebastian@seattleu.edu), Yong Long Tan (ytan@seattleu.edu)
Faculty Advisor: Dr. Wan Bae
Sponsor: Dr. Sada Narayanappa @ Geospaces.org

Abstract:

Around 83% of employees spend a third of their workweek attending meetings. We aim to conduct analysis from speech to text specifically in conversations with more than two speakers in a meeting environment. Besides granting accessibility accommodations to hard-of-hearing people, we also want to provide accurate and insightful analysis to customers and researchers. With recent advances in automatic speech recognition and natural language processing models such as OpenAI's Whisper and Meta's BART, we seek to simplify the process of speech recognition and abstractive summarization of long meetings. Unlike extractive summarization, abstractive summarization creates summaries by synthesizing new words and sentences that maintain the original meaning of the source. This is challenging due to the necessity of NLP models for text generation. Our solution consists of developing a pipeline that takes large audio files (approximately 1 hour) and transcribes and diarizes the conversation using OpenAI's Whisper and Pyannote. Our summarization solution involves retraining the BART model. Our retrained model makes the base-model compatible with summarizing long meeting dialogues and improves summarization by 148%. The solution includes a front-end website built using Django that integrates our model and enables users to upload a meeting recording to get the full meeting transcript and summary.

1. Introduction

1.1. Background

Meetings with multiple speakers can be challenging for individuals who require accessibility accommodations, such as hard-of-hearing people. Audio-to-text conversion has been a helpful tool to assist these individuals. Still, it does not address the issue of identifying the speaker in a meeting and generating a summary for the given meeting. Therefore, there is a need for a pipeline that can identify the speaker and summarize a given audio file. Besides granting accessibility accommodations to hard-of-hearing people, we also want to provide accurate and insightful analysis to customers and researchers, which can help them understand and utilize the content of the meetings better. In addition to providing accessibility accommodations for individuals who are hard of hearing, we want to offer precise and insightful analysis to both customers and researchers. This analysis can improve their comprehension and utilization of meeting content. This can be especially beneficial for organizations that need to monitor multiple meetings and extract key information for decision-making and research purposes.

1.2. Problem Statement

We aim to conduct audio analysis from speech to text specifically in conversations with more than two speakers in a meeting environment. Besides granting accessibility accommodations to hard-of-hearing people, we also want to provide an accurate and insightful analysis for customers and researchers to leverage.

This audio analysis problem can be partitioned into four different parts:

i. **Automatic Transcription** — we used OpenAI's Whisper to automatically transcribe audio files.

ii. **Speaker Diarization** — we used an opensource Python library called Pyannote to perform basic speaker diarization.
iii. **Summarization** – we used a pre trained BART model trained on the CNN/Dailymail and SAMSUM datasets.
iv. **Visualization/Presentation** — we developed a basic front-end web application where users can upload an audio file and receive a transcript and summary.

1.3. Our Contribution

We created a working summarization pipeline starting with an Automatic Speaker Recognition (ASR) module consisting of OpenAI Whisper and Pyannote. For our summarization module, we performed transfer-learning on a pre trained BART sequence-to-sequence transformer model by training the model on a machine transcribed AMI/ICSI corpus to improve summarization for long meetings.

2. Related Work

2.1. In recent years, the transformer encoder-decoder architecture has been widely used in the field of Natural Language Processing (NLP). Some related work include:

i. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension: BART is pretrained by corrupting the input text using some noise schemes and learning a model to reconstruct the original input. It reads the corrupted input in both directions, left to right or vice versa. The bidirectional encoder produces a set of hidden states that capture the meaning and context of our input text. Then, this collection of hidden states will get pushed to the autoregressive decoder. The decoder is another component that generates the output text one element at a time, where each element is conditioned on the previously generated elements.
ii. QMSum: A New Benchmark for Query-based Multi-domain Meeting Summarization
iii. pyannote-whisper: An open source diarization library that helps with speaker diarization where the audio is partitioned into segments to identify speakers.

2.2. In parallel with the Literature survey, we also identified tools that helped us to prepare the pipeline that we need to construct for the project. Some of the tools we used to make out pipeline include:

i. Whisper - https://github.com/openai/whisper
ii. Pyannote - Partitioning speakers (diarization) - Amazon Transcribe

3. Methods

3.1. Overview of proposed architecture

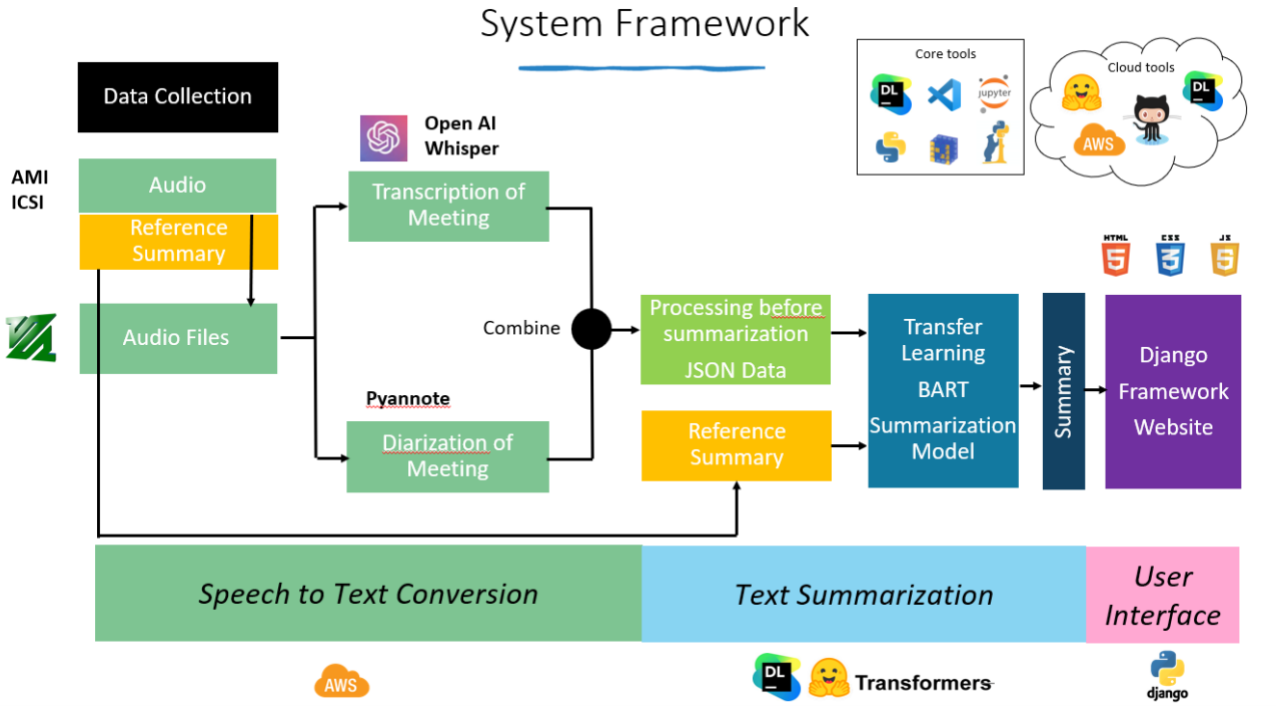

Figure 1. High Level Architecture of Parrot

Our architecture can be separated into three distinct parts.

i. Speech to Text Conversion: Using the AMI and ICSI corpus, we transcribed the audio files using Whisper. Next, we diarized the audio file using Pyannote and combined the two files into a single transcription.

ii. Text Summarization: Using the human reference and the dataset that we have created we combined both to form a single JSON file. We then fed the JSON file into the pretrained BART model.

iii. User Interface: Using the Django Framework, we created a simple web application that allows users to upload an audio file and receive a transcription and a summary of the meeting.

The complete List of tools we used is as follow: Python 3.8, Jupyter Notebook, VSCode, Datalore, FFMEG, Whisper Open AI, Pyannote, PyTorch, CUDA, Hugging Face Transformers, AWS, BART – Model, Github, Django, HTML, CSS, Javascript. To setup the project environment please refer to project_setup.md file on Github which has instructions to setup all the dependencies for our project.

3.2. Proposed Method and Algorithm

Our pretrained model is BART (Bidirectional Auto-Regressively Transformer). BART was developed by Facebook AI Research team. BART can handle various NLP tasks, for example, translation, text classification, summarization, etc. BART can be fine-tuned for any text generation and comprehension tasks. In short, BART is pretrained by corrupting the input text using some noise schemes and learning a model to reconstruct the original input.

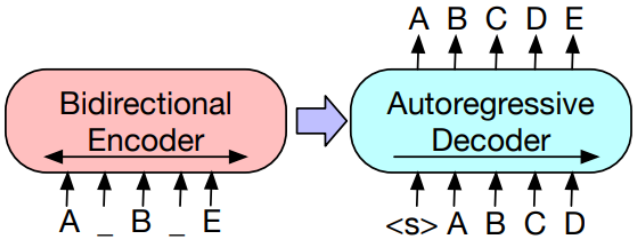

Figure 2. Encoder Decoder Model

BART has three big components: 1) noise transformation, 2) bidirectional encoder, and 3) autoregressive decoder.

i. Noise transformation happens right before the encoder. Noise transformation will add some noises onto the inputs before feeding the data into the encoder. Having such action is to prevent the overfitting problem. There are five ways of corrupting the input: Token Masking, Sentence Permutation, Document Rotation, Token Deletion, and Text Infilling. The authors examined all of them and found that BART with text infilling could give the most consistently strong performance.

ii. Next, the corrupted data will feed into the encoder that is able to read in both directions. This encoder produces a set of hidden states that capture the meaning and context of the input text.

iii. Lastly, the set of hidden states will then get pushed to the autoregressive decoder that reads from left to right. This decoder basically generates a probability distribution over all possible tokens, given the hidden states and the masked input text. It then samples a token from the distribution and appends it to the output text in a step-by-step manner, where each token is conditioned on the previously generated tokens.

One reason why we choose BART is that it has richer documentation on how users can do fine-tuning and feed their own dataset to suit their own needs, compared to other novel pretrained models. Also, BART can establish abstractive summarization which is our objective. The evidence shown in the paper is highly abstractive with only a few phrases copied from the input.

4. Experiments

4.1. Datasets

i. **Data Sources:**
AMI Corpus, Augmented Multi-party Interaction Project, funded by the European Commission - https://groups.inf.ed.ac.uk/ami/corpus/
ICSI Corpus, International Computer Science Institute Speech Group, UC Berkley - https://groups.inf.ed.ac.uk/ami/icsi/
Reference Summary - Yale-LILY/QMSum: Dataset for NAACL 2021 paper: "QMSum: A New Benchmark for Query-based Multi-domain Meeting Summarization" (github.com)

ii. **Details of the Datasets**
AMI corpus and ICSI corpus are publicly available datasets that contain 100 hours and 70 hours of meeting conversations recorded respectively. The duration of the audio files is between 20 minutes to 1 hour and 30 mins and there are 169 files in total. All the files are in .wav format. The audio files consist of recording of multiple speakers interacting in workplace meeting.
We are also using reference summaries to train the model. The reference summaries are made by humans for the same meetings found in the AMI and ICSI corpus and are in text-format. The summaries are around a paragraph in length with 80 to 150 words.

iii. **Characteristics of the Datasets**
The audio files are relatively clean but quite lengthy, each conversation is around 50 mins long. One difficulty that we are facing is the length of the audio files. We are relying on OpenAI's Whisper to transcribe the audio files and also using pyannote to diarize the sudio before combine the two to create the complete transcript. We used 1-GPU for carrying out the processing on the audio files. It takes approximately 3 mins. With 1-GPU to generate a transcript for a audio file that is 50 mins to 1 hour long.

4.2. Exploratory Data Analysis Results

Because we are dealing with NLP, we do not perform traditional data analysis. However, because the format of our dataset was not suitable for Summarization tasks, we had to perform data wrangling procedures.

i. **Errors in Transcription using Whisper**
Due to the fact that we did not specify a specific language for the Whisper model, it was inferring languages from accents. This caused several errors in the transcriptions where Whisper switched to another language before reverting to English.

ii. **Data Wrangling**
A typical dataset for summarization will have at least three columns: id, original text, and reference summary. However, since our dataset is a transcription of audio corpuses, we had to find reference human summaries. Fortunately, we were able to find reference summaries for the ICSI and AMI datasets.

Next, we had to combine all three features together so that they will be readable by HuggingFace's Dataset API. We created a Python function to combine the meeting id, dialogue, and reference summary together and wrote them to a single JSON file. In other words, each JSON object is an individual meeting.

```
DatasetDict({
    train: Dataset({
        features: ['meeting_id', 'summary', 'dialogue'],
        num_rows: 156
    })
    test: Dataset({
        features: ['meeting_id', 'summary', 'dialogue'],
        num_rows: 40
    })
})
```

Figure 3. JSON format

4.3. Experiment Setup

As mentioned above, BART hasn't been trained on any long conversation dataset that includes multiple speakers. We plan to feed some of the AMI and ICIS corpus to BART and utilize another portion of the data to validate the outcome.

We are using transfer learning to train our model. In transfer Learning we need select a baseline model which acts as the source model, and we fine-tuned using the dataset. For each of the experiments we evaluated the experiment outcomes using both qualitative and quantitative metrics which are discussed in the nest section.

For faster prototyping, we used the HuggingFace Trainer API which allows us to use train models and save the best. The Trainer API also allows us to use TensorBoard to evaluate the training and evaluation performances. Once uploaded to HuggingFace, we can utilize the instant Inference API to use our model for demos.

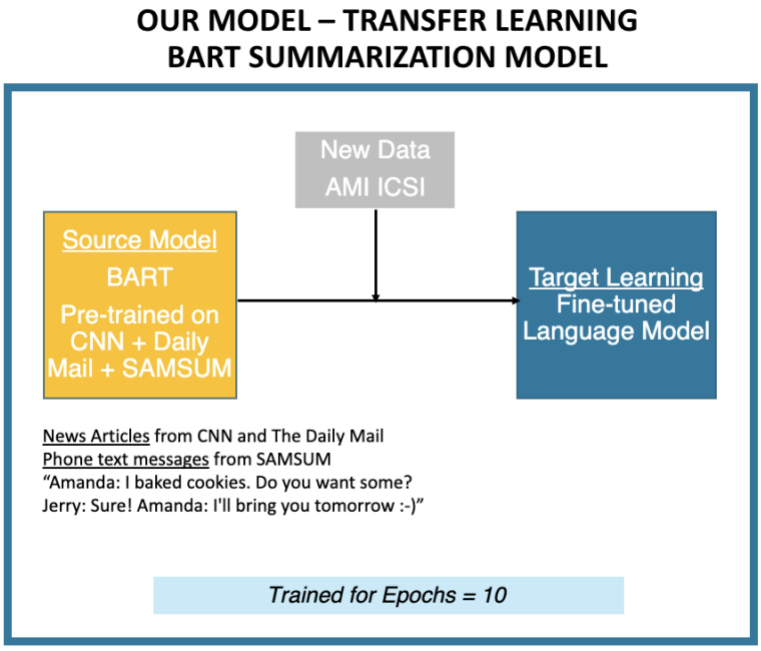

Figure 4. Our Transfer Learning Model

The following are some of the experiments that we setup:

i. Changing the baseline model
   - We attempted to use a baseline model without the pretraining on SAMSUM. However, the summary that the model produced was unsatisfactory for our purposes.

ii. Different number of training epochs:
   - We attempted to increase the number of epochs to ten for the second model. However, after the first epoch, the validation loss continues to increase and our HuggingFace model automatically saved the model from the first epoch.

iii. Some of our experimental models can be found at:
   - https://huggingface.co/vmarklynn/bart-large-cnn-ami-icsi
   - https://huggingface.co/vmarklynn/bart-large-cnn-icsi-ami-v3

4.4. Evaluation Metrics

Our main evaluation metric will be ROUGE (Recall-Oriented Understudy for Gisting Evaluations). It compares automatically produced summaries against reference summaries which are human produced.

For example, if a machine summary produced "the cat was found under the bed," it would be compared against "the cat was under the bed." One problem that ROUGE has is that the reference summaries need to be done by humans. However, we have found previous human summaries for our data set as mentioned above in the dataset section.

ROUGE measures the following: Recall: how much of the reference summary is recovered from the reference? Precision: how much of the system summary was relevant? The Following performance metrics were used in experiments.

i. **Rouge-N**: This measures the overlap between the generated text and the reference text in terms of n-grams. To compute the Rouge-N score,
   - Tokenize the generated text and the reference text into n-grams of size N.
   - Compute the number of overlapping n-grams between the generated text and the reference text.
   - Compute the precision, recall, and F1-score using the following formulas:
     - ✓ Precision = Number of Overlapping N-grams / Number of Tokens in the Generated Text
     - ✓ Recall = Number of Overlapping N-grams / Number of Tokens in the Reference Text
     - ✓ F1-score = (2 * Precision * Recall) / (Precision + Recall)

ii. **Rouge-L:** This measures the overlap between the generated text and the reference text in terms of longest common subsequences. To compute the Rouge-L score,
   - Tokenize the generated text and the reference text into individual words.
   - Compute the longest common subsequence between the generated text and the reference text.
   - Compute the precision, recall, and F1-score using the following formulas:
     - ✓ Precision = the calculated LCS / Number of Tokens in the Generated Text
     - ✓ Recall = the calculated LCS / Number of Tokens in the Reference Text
     - ✓ F1-score = (2 * Precision * Recall) / (Precision + Recall)

iii. **Rouge-Lsum**: It computes the Rouge-L score for multiple sentences, and then averages the scores. In other words, Rouge-Lsum is the same as Rouge-L, but it is calculated at the document level, considering all sentences together, rather than calculating it per sentence and then averaging the scores.

iv. **Gen Len** (Generated Length): This measures the average length of the model-generated sequences. It is used to evaluate how well a language generation model can produce output of the desired length.

A good Rouge-1 score typically falls between 40 and 50, while a Rouge-L score of around 30 or higher is generally considered good. However, these metrics have limitations as they primarily focus on the syntactical overlap and disregard semantic aspects. For instance, if the model effectively rewords and restructures the original text, Rouge scores might not fully capture its performance. To overcome these limitations, it is beneficial to involve multiple experts with diverse backgrounds who are skilled at summarizing. By incorporating their feedback along with the performance metrics, the quality of the output generated by the model can be significantly enhanced. This approach offers a more comprehensive evaluation that considers both syntactical and semantic aspects of summarization.

4.5. Evaluation Results and Analysis

For our first finetuning task, we finetuned the BART model that was trained on the CNN Daily Mail and SAMSUM dataset. Although this model is suitable for summarization tasks where we are summarizing articles and short chat messages, it is not suitable for summarizing long dialogues. We finetuned the model using HuggingFace's Trainer API with the following hyperparameters:

i. learning_rate: 5e-05
ii. train_batch_size: 8
iii. eval_batch_size: 8
iv. seed: 42
v. optimizer: Adam with betas=(0.9,0.999) and epsilon=1e-08
vi. lr_scheduler_type: linear
vii. num_epochs: 5

Our quantitative results throughout the epochs are shown in the table below:

Training Loss	Epoch	Step	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
No log	1.0	20	3.2095	39.8174	11.5559	24.0296	36.3048	108.5
No log	2.0	40	3.1361	39.7563	11.1286	23.2632	36.5664	108.15
No log	3.0	60	3.1599	41.79	12.0967	23.5336	37.6859	122.95
No log	4.0	80	3.2878	42.3161	12.2801	23.9352	38.2391	122.7
No log	5.0	100	3.3671	40.7968	10.7336	22.9434	36.4383	129.225

Figure 5. Initial results from our first finetuning

On our second finetuning, we increased the epochs to ten. Although our ROUGE-N scores were generally lower than the first, we felt that the second model gave better summarizations than the first; therefore, we decided to use the second iteration of the model for our demonstrations.

The second model was trained using the following hyperparameters:

a. learning_rate: 5e-05
b. train_batch_size: 1
c. eval_batch_size: 1
d. seed: 42
e. optimizer: Adam with betas=(0.9,0.999) and epsilon=1e-08
f. lr_scheduler_type: linear
g. num_epochs: 10

Training Loss	Epoch	Step	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
No log	1.0	135	3.2308	38.4274	13.6461	22.366	35.2353	185.7941
No log	2.0	270	3.3026	40.1748	11.7944	23.2655	36.4718	146.8529
No log	3.0	405	3.5199	39.8209	12.1621	22.7772	36.4967	141.7647
2.2131	4.0	540	4.0508	40.4325	11.6547	22.9958	36.8782	131.4412
2.2131	5.0	675	4.6988	38.4097	9.8309	20.3894	34.1967	145.9706
2.2131	6.0	810	4.9590	38.5758	9.6335	20.865	35.0321	169.2353
2.2131	7.0	945	5.4264	38.2813	9.5764	21.1406	34.5989	148.0294
0.401	8.0	1080	5.4887	38.3014	9.6881	21.2398	34.1584	139.3529
0.401	9.0	1215	5.8044	39.9603	10.4329	22.6895	36.2406	145.2353
0.401	10.0	1350	5.8987	39.0928	10.8408	21.9138	35.5067	138.7941

Figure 6. Results from our second finetuning

The following are some qualitative results using the second model:

As seen below, compared to the human reference summary, the output from our model is more comprehensive, accurately identifying the specific locations visited by the individuals.

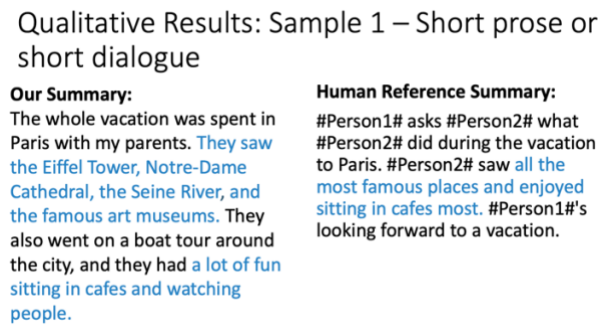

Figure 7. Short prose or dialogue comparison

We tested the limits of our model by giving input of the entire Lord of the Rings screenplay to summarize. It has been able to provide a reasonable summary but as one can see it makes a wrong inference that people should be more like the KINGS of Men when they shouldn't because the KINGS of Men become corrupted by power. We can also see that the model has learned typical workplace lingo such as "The meeting began …" and "They discussed …" due to the transfer learning and finetuning we have carried out.

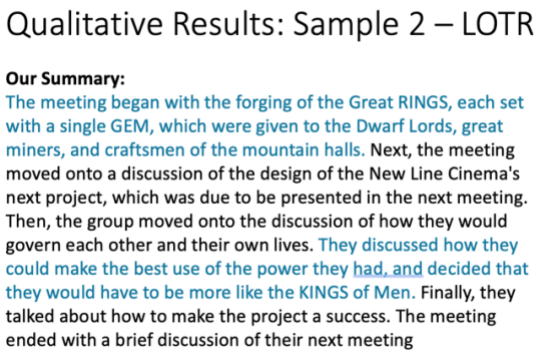

Figure 8. Summarization of Lord of the Rings complete screenplay

Our model covers about 60 percent of the content of the reference summary and performs well in most parts. However, in the second sentence, the phrase 'project manager' is unnecessarily repeated at the end.

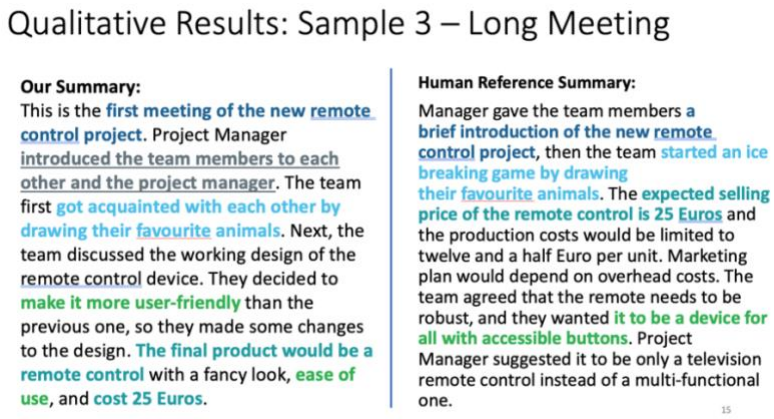

Figure 9. Long meeting summarization comparison

4.6. Public Links to the Project

i. GitHub Link - https://github.com/vmarklynn/parrot

ii. Link to Models on Hugging Face:

- https://huggingface.co/vmarklynn/bart-large-cnn-samsum-icsi-ami
- https://huggingface.co/vmarklynn/bart-large-cnn-samsum-icsi-ami-v2

iii. Link to demo on YouTube -https://youtu.be/3fHrQ8I0j4c

iv. Link to presentation on YouTube – https://youtu.be/QHgWTdenAyM

5. Conclusion:

We introduced a pipeline that transcribes, diarizes, and summarizes conversation meetings using Whisper, Pyannote, and a retrained BART model, resulting in a 148% improvement in summarization. Our solution is integrated into a user-friendly website built with Django, allowing users to upload meeting recordings and obtain comprehensive transcripts and summaries. Our work paves the way for more effective meeting analysis and information accessibility in various contexts and industries.

However, our project currently faces limitations in terms of the quality of the human reference and the availability of data. Expanding the scope and performance of our summarizer to cover different meeting domains requires access to more diverse and comprehensive data sets. This would enable the model to learn and adapt to various meeting scenarios, ultimately improving its accuracy and generalizability.

Future work should focus on making our application more marketable to companies by improving the summarization capabilities and providing accurate and insightful analysis of workplace meetings. Additional application features could include visualizations of meeting using colors and waveforms, speaker recognition that allows the model to learn overtime to recognize previously encountered voices, and sentiment analysis.

6. References

i. Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer, “BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension,” 2019, https://arxiv.org/pdf/1910.13461.pdf

ii. Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, and Dragomir Radev, “QMSum: A New Benchmark for Query-based Multi-domain Meeting Summarization,” 2021, https://aclanthology.org/2021.naacl-main.472.pdf

iii. Mccowan, Iain & Carletta, J & Kraaij, Wessel & Ashby, Simone & Bourban, S & Flynn, M & Guillemot, M & Hain, Thomas & Kadlec, J & Karaiskos, V & Kronenthal, M & Lathoud, Guillaume & Lincoln, Mike & Lisowska Masson, Agnes & Post, Wilfried & Reidsma, Dennis & Wellner, P. (2005). The AMI meeting corpus. Int'l. Conf. on Methods and Techniques in Behavioral Research.

iv. A. Janin et al., “The ICSI Meeting Corpus.” 2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03)., Hong Kong, China, 2003, pp. I-I, doi: 10.1109/ICASSP.2003.1198793.

v. Radford, Alec & Kim, Jong & Xu, Tao & Brockman, Greg & McLeavey, Christine & Sutskever, Ilya. (2022). Robust Speech Recognition via Large-Scale Weak Supervision.

vi. H. Bredin et al., "Pyannote.Audio: Neural Building Blocks for Speaker Diarization," ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 2020, pp. 7124-7128, doi: 10.1109/ICASSP40776.2020.9052974.

vii. https://github.com/yinruiqing/pyannote-whisper

7. Appendix

7.1. Data Science Methodology

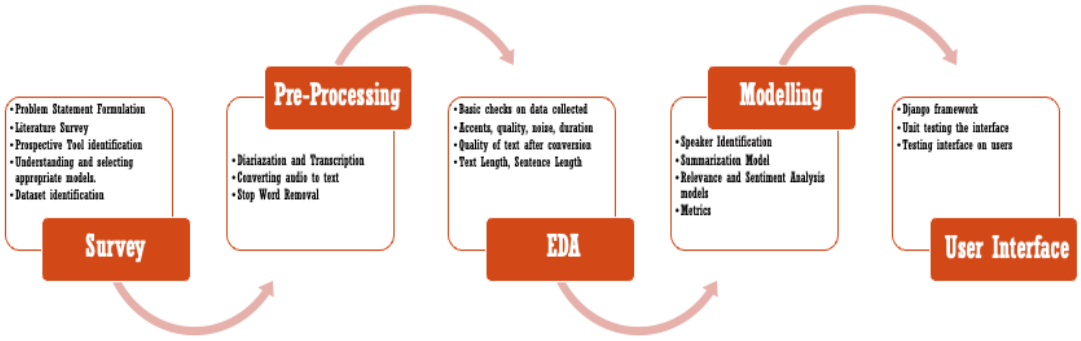

Figure 10. Data Science Diagram

Our pipeline begins with transcribing the test data that we will collate into individual text files using Whisper. We will asl use *pyannote.audio* (pyannote/speaker-diarization · Hugging Face) to diarize the audio. Next, we will perform data preprocessing with the transcribed texts. This includes lemmatization, stop word removal, and lower casing using the *NLTK* package in Python.

Currently, we are planning to use the *transformers* Python package to test out various encoder-decoder models including the BART model above. This is because the *transformers* package contains pretrained encoder-decoder models that will allow us to produce our deliverables for the project. Although the models are pretrained, we will fine tune the models as appropriate. Finally, to analyze how our pretrained model performed, we will be using the ROUGE metric—a commonly used metric to evaluate summarizations.

7.2. Evaluation Metrics for the Project

For any project it is important to balance the Triple Contraint between Time-Scope-Quality. This is going to be a challenge since for us the development time is fixed to a 6-week period over which we must complete the entire development part of the project. In order to ensure that the project runs smoothly, and we can have on-time deliverables, we feel that it is important to track our activity and use an agile framework.

We plan to develop the product or complete the project incrementally by using teams to track our sprint activities and have weekly sprints. This way we can objectively review our performance as a team by looking at how many tasks we have been able to complete.

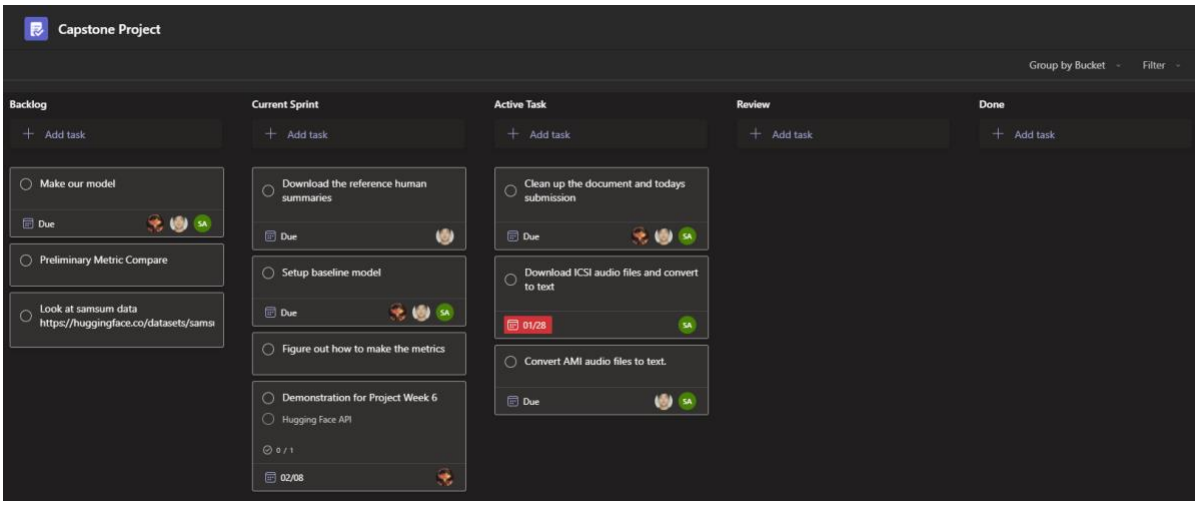

Figure 11. Kanban Task Board

We plan to maintain quality by ensuring that the project and the product that is developed through this project is useful and easy to use. We want the interface to be easy and simple and we plan on asking our classmates to use the interface and incorporate their feedback. For the tool to be useful it should be fast we are targeting that the summary should be generated within 5 mins for a 50 min meeting conversation.

The scope for this project is quite tight since we are on a fixed time schedule. We currently have an expanded scope and we have narrowed the scope down to the most critical components as stated in the updated scope section above. We first target on creating a MVP and then incrementally adding features.

7.3. Early UI sketches:

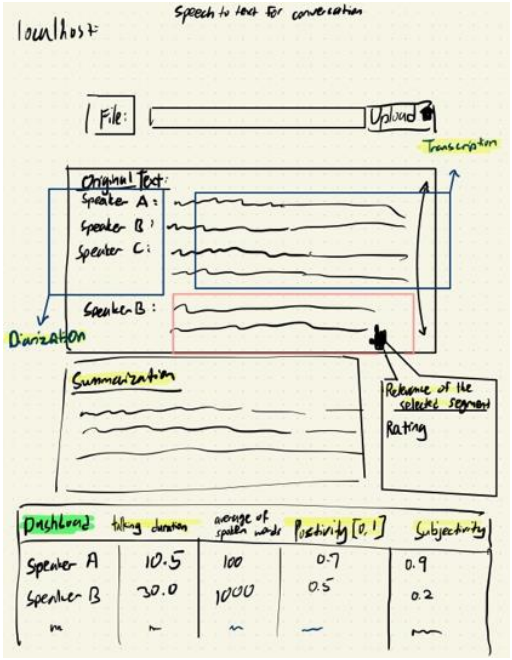

Figure 12. UI Diagram