**Patient - Nurse Audio call Transcript Generation:**

**What I have done:**

I have built the code ‘v1\_pr.py’ that leverages the pre-trained Wav2Vec2 model and processor from the "facebook/wav2vec2-base-960h" checkpoint to transcribe audio. It begins by loading an audio file with the librosa library and employs the noisereduce library to enhance audio quality. Next, the Wav2Vec2 processor prepares the audio for the model. The model then generates logits, which are converted into predicted ids using argmax. Finally, the processor's batch\_decode function transforms the ids into the transcribed text output.  
  
**What I tried to do:**

I built the same code ‘v1\_fitu.py’ but added even more functionalities by fine-tuning the pre-trained Wav2Vec2 model on our local custom dataset (if provided in future). It prepares the dataset, creates an AudioDataset for loading audio and transcriptions, and implements a training loop. During the loop, the model is iteratively optimized using your data, with the optimizer adjusting weights based on the calculated loss. This fine-tuning process tailors the model to the specific characteristics of your dataset, potentially leading to more accurate transcriptions. Finally, the fine-tuned model and processor are saved for future use. Given the right amount of time and computational power, I can very much run this code.

**What I intend to do overtime:**

I intend to perform this particular use case using the model called Conformer. If given time to research and implement, I'll first need to modify my codebase. I'll start by switching the import statements to use Conformer modules from Hugging Face Transformers—probably something like from transformers import ConformerForCTC, ConformerProcessor. I'll likely experiment with a suitable checkpoint like "facebook/conformer-ctc-small" as a base. Since the core concepts of data loading and preprocessing remain similar, my AudioDataset class might only need minor adjustments to work with the Conformer processor.

The most significant shift will be in the training loop. Conformer's unique architecture, combining convolution and self-attention, could necessitate tweaks in the learning rate or optimizer settings to ensure optimal convergence. I might also investigate techniques like SpecAugment (data augmentation specifically for speech) to further bolster the model's performance.

I anticipate the Conformer model potentially surpassing Wav2Vec2 in transcription accuracy. Its ability to model long-range dependencies in audio should prove especially beneficial for handling complex or noisy audio samples. Additionally, Conformer's architectural design may translate to computational efficiency gains, although the trade-off could be longer fine-tuning times or a need for a larger dataset to fully capitalize on its advantages.

**Redaction of Transcript text:**

**What I have done:**

In the file ‘v2\_pr.py’, I loaded the "en\_core\_web\_md" SpaCy NLP model for entity recognition. I developed the redact\_personal\_info function, which uses SpaCy's NER capabilities to identify names, locations, dates, and organizations, replacing them with '██████'. For even better coverage, I incorporated regular expressions to redact various phone number formats, email addresses, and potential medical record numbers. Finally, I ensured the redacted text was securely saved to an output file.

**What I intend to do:**

I would like to integrate this redaction process with the previous transcription module that would have used the trained Conformer model and also create a custom dataset on more unobvious personal information and apply boosted ML algorithms in a pipeline and apply them to identify more unique personal information texts to redact in the final flow of the process.

**Generate a summary of the transcript:**

**What I have tried and done:**

In the file ‘v3.py’, I loaded the 'ccdv/pubmed-summarization' dataset from Hugging Face for scientific article summarization. After some data cleaning, I used Blurr's SummarizationPreprocessor to prepare the text for the "google/pegasus-large" model, ensuring length constraints were appropriate for input and output. I obtained the Pegasus architecture, configuration, tokenizer, and model from Hugging Face Transformers. I used Blurr's Seq2SeqTextBlock and Seq2SeqBatchTokenizeTransform to ensure proper dataset tokenization and formatting for model compatibility. A DataBlock was created to manage the dataset flow, a RandomSplitter to divide the data, and data loaders prepared for training. I included comprehensive metrics (ROUGE, BERTScore, BLEU, and more) for evaluation. Blurr's Learner wrapped the model, and I selected a loss function, optimizer, and callbacks. After finding a suitable learning rate, I trained the model for 10 epochs, observing the defined metrics. Finally, I fine-tuned the model with full precision and exported my trained Pegasus summarizer conveniently as "pegasus\_summary\_export.pkl".

**What I intend to do overtime:**

Given the time and right computational power, I can also load our custom dataset to finetune this model for our dataset which I had done in the ‘v1\_fitu.py’ file. We can also make this particular model as a callable local API to be used to generate chats in the places where we might integrate chatbots in the portal dashboard used by the hospital staffs and Doctors as Pegasus model is very powerful and if we tap on its model weights and trainable variables, we can make it a foundational code to be used anywhere for text generation and conversational use cases.

**Report generation for Patient and Doctor’s View:**

**What I have done:**

In the file ‘v4.py’ imported the AutoModelForCausalLM and AutoTokenizer classes from Hugging Face Transformers to work with my language model. I loaded my pre-trained AlpaCare LLM (e.g., "xz97/AlpaCare-llama1-7b"), its associated tokenizer, and ensured they were on the appropriate device (GPU or CPU). I tokenized the input summary with the tokenizer, taking care of padding, truncation, and length limits for optimal model input. Finally, I employed the model's generate method, adjusting parameters for output length and the number of medication/precaution suggestions to be generated.

**What I intend to do overtime:**

As usual, I want to load our custom dataset and fine tune the already developed model to perform and execute custom use case related functions and provide results. I also want to integrate this AlpaCare LLM model with the previously used Pegasus model to generate great results and accuracy.

**Integration of BB84 Quantum protocol with the Transcript text:**

I had previously developed an application using Python and Flask which serves as a tool to use the BB84 protocol with qiskit Library to encrypt and decrypt messages sent between 2 Users. I had developed this project before two to three months ago as part of my Course work in the first semester. I intend to integrate this protocol with this project in every step of data getting transferred ie. every module when the transcribed text is transferred to the next module for another content generation.

The Python and Flask application I've developed using the ‘BB84.py’ & ‘quantum\_utils.py’, which utilizes the Qiskit library to implement the BB84 protocol, represents a sophisticated quantum cryptographic system aimed at securing communications between users. This advanced system, created during my initial semester coursework, integrates complex quantum mechanics principles to encrypt and decrypt data, thus providing a level of security that is currently unparalleled. Through quantum\_utils.py, the application elaborates on quantum circuits, exploiting qubit polarization and measurement to enable quantum key distribution effectively. These critical operations form the backbone of a quantum-secured communication channel, meticulously preserving data integrity and confidentiality throughout the application's architecture. By strategically deploying quantum encryption and decryption across various modules responsible for processing and transferring transcribed text for further content development, the project not only showcases the practical application of quantum cryptography but also sets a new benchmark for secure digital communication. This ensures that every piece of text, as it moves from one module to the next within the project, is enveloped in a quantum-secure layer, making unauthorized access computationally infeasible and safeguarding the sanctity of sensitive information.

**Calendar Scheduling:**

**What I have tried and done:**

For this module I propose we use a method called Federated Learning (FL). I will start by selecting TensorFlow Federated (TFF) or PySyft as the core framework for the federated learning (FL) component of my system, enabling the training of machine learning models on decentralized data. This approach will involve designing a predictive model that incorporates variables such as the doctor's availability, patient preferences, and the urgency level indicated by the call transcriptions.   
  
For now in the file ‘FL.py’, I have developed an algorithm that simulates the working logic and concepts of a fully trained Federated Learning model which I can develop with more time and computation power. In the simulation I've constructed, the essence of Federated Learning (FL) is captured through a decentralized approach to scheduling healthcare appointments. This simulation is built using Python and NumPy, showcasing a scenario where doctor and patient data, such as availability and appointment needs, are randomly generated to reflect the diverse and dynamic nature of a healthcare environment. The simulation includes mechanisms for mapping patient severity to a numerical priority, introducing differential privacy through noise addition to protect privacy, and aggregating updates in a privacy-preserving manner—key components that mirror the FL process of local computation and global aggregation without compromising data privacy.

The core of this simulation lies in its ability to demonstrate the federated learning cycle through the selection of subsets of doctors and patients, simulating local improvements in doctor availability, and aggregating patient needs with privacy considerations. For instance, the select\_clients function represents the FL concept of selecting random subsets of nodes (in this case, doctors and patients) for each learning cycle. The noise\_for\_privacy and aggregate\_with\_privacy functions simulate the application of differential privacy, a technique vital for ensuring that the aggregated data used to update the global model does not reveal sensitive information about the participants. The simulation advances through cycles, with each cycle representing an iteration of federated learning where local updates are aggregated to improve the global scheduling model. The effectiveness of the scheduling is evaluated based on the match rate between patient needs and doctor availability, reflecting the goal of optimizing resource allocation in a real-world federated learning scenario.

**Portal and dashboard creation:**

The envisioned healthcare dashboard, to be developed using Flask and potentially Django for broader web capabilities, will feature advanced technical implementations for a secure and interactive user experience. User authentication will be managed via Flask-Login and Flask-JWT-Extended, ensuring robust security for Register and Login functionalities. Once authenticated, doctors will access a dashboard populated with Patient IDs, each clickable to reveal a Generated Summary Report, related medical literature, precautionary reports, risk assessments, and appointment statuses. This functionality will be powered by integrating the "google/pegasus-large" NLP model for summary generation and Elasticsearch for efficient medical literature retrieval, alongside an API call to databases like PubMed for related references.

The dashboard's backend will harness Flask-SocketIO for real-time updates on patient appointment acceptance, while the precautionary reports and risk levels will be generated using custom models, including the "xz97/AlpaCare-llama1-7b" for precautionary measures and TensorFlow Federated for decentralized, privacy-preserving predictive analytics. This comprehensive integration of Flask, Django, JWT authentication, Hugging Face's Transformers, Fastai, Elasticsearch, and TensorFlow Federated encapsulates the project's commitment to leveraging cutting-edge technologies for enhancing healthcare management and decision-making processes, ensuring a secure, efficient, and user-centric platform for doctors. The Federated Learning model will be used to link each patient to the available doctor to be viewed in the dashboard.