1. Nurse - Patient Audio call transcription:
2. What I have done:

Model Loading: It first loads the pre-trained Wav2Vec2 processor and model from the "facebook/wav2vec2-base-960h" checkpoint. The processor is responsible for preprocessing audio inputs, while the model is used for making predictions.

Audio Processing: The code then loads an audio file using the librosa library and applies noise reduction using the noisereduce library. This step is crucial for improving the quality of the input audio, which can lead to more accurate transcriptions.

Transcription: After processing the audio, the code uses the Wav2Vec2 processor to encode the audio into input values suitable for the model. It then passes these input values to the model to obtain logits, which are converted to predicted ids using argmax. Finally, the predicted ids are decoded back into text using the processor's batch\_decode method, resulting in the transcription of the audio file.

1. What I Intend to do over time:

Fine-tuning the Wav2Vec2 Model: Use the Hugging Face Transformers library to fine-tune the pre-trained Wav2Vec2 model on your custom dataset. This involves defining a data processing pipeline that tokenizes the audio files and transcriptions, along with training parameters such as batch size, learning rate, and the number of training epochs.

Conformer Model Architecture: Instead of using the base Wav2Vec2 model, you can leverage the Conformer model architecture for transcription. The Conformer model is a transformer-based architecture that integrates depthwise convolution and self-attention mechanisms, making it well-suited for capturing long-range dependencies in audio data.

Training Process: During training, the model learns to map input audio features to tokenized text sequences. The Conformer architecture's self-attention mechanism allows it to capture contextual information from the audio, potentially leading to more accurate transcriptions compared to the base Wav2Vec2 model.

Inference: Once the model is trained, you can use it to transcribe audio calls from patients to nurses. The inference process involves loading the trained Conformer model and its corresponding processor, processing the audio input, and decoding the model's output to obtain the transcriptions.

2. Encrypt the transcribed text using the BB84 Quantum protocol:

We would use a BB84 Quantum protocol based on qiskit package from IBM to encrypt and as well as decrypt the transcribed text of the patient to make it secure within the walls of this automated project.

1. Secure Communication: The BB84 protocol can be used to establish a secure communication channel between the nurse (Bob) and the patient (Alice). This ensures that their conversation remains confidential and protected from eavesdropping.
2. Quantum Encryption: By encrypting the transcription process using the BB84 protocol, sensitive medical information discussed during the nurse-patient call is protected. The use of quantum encryption adds an extra layer of security, ensuring that only authorized parties can access the information.

3. Redact Personal Information:

1. What I have done:

Loading Spacy NLP Model: The code first loads the SpaCy NLP model "en\_core\_web\_md," which is a medium-sized English language model trained on web text. SpaCy is a natural language processing library used for tasks such as named entity recognition (NER).

Redacting Personal Information: The redact\_personal\_info function takes two arguments: the path to the input file containing the transcribed text and the path to the output file where the redacted text will be saved. It reads the input file and uses SpaCy's named entity recognition (NER) capabilities to identify entities like names (PERSON), locations (GPE), dates (DATE), and organizations (ORG). It replaces these entities with the string '██████', effectively redacting them from the text.

Enhanced Redaction for Phone Numbers: The code uses a regular expression to find and redact phone numbers in various formats (e.g., 123-456-7890, 123.456.7890). It replaces them with '██████'.

Redacting Email Addresses: Another regular expression is used to find and redact email addresses, replacing them with '██████'.

Redacting Medical Record Numbers: The code uses regular expressions to redact medical record numbers. It covers patterns like "medical record number is 12345" and standalone numbers of 5 digits or more. These numbers are replaced with '██████'.

Saving the Redacted Data: Finally, the redacted text is saved to the output file specified.

1. What I intend to do overtime:

Model Loading: You would start by loading your fine-tuned Conformer model, which is trained to transcribe audio recordings of nurse-patient calls into text. This model is crucial for accurately converting audio inputs into textual transcriptions.

Transcription Process: Once the model is loaded, you would use it to transcribe the audio recordings of nurse-patient calls. This step remains the same as your previous implementation, where the Conformer model processes the audio inputs and produces text transcriptions.

Redaction Integration: After obtaining the transcribed text, you would integrate the redaction process into your workflow. This involves applying the redact\_personal\_info function to the transcribed text, identifying and redacting personal information such as names, locations, dates, and phone numbers.

Redacted Transcriptions: The redacted transcriptions are then saved or processed further, ensuring that any sensitive information is protected. This step is crucial for maintaining patient privacy and complying with healthcare regulations.

4. Generate a summary of the transcript for the Doctor:

Here we will use the transcript to generate a detailed third person POV for the doctor to read and analyze using a bigger model called Pegasus which we will use to train and fine tune our datasets.

I have developed the model. In future I would run the model on powerful systems to fine tune as well as train and fit our local custom dataset to provide very brief and customized results unique to our application and use case. To build the model I have followed the below steps:

Dataset Loading: The code loads a dataset containing PubMed articles and their corresponding abstracts using the Hugging Face dataset library.

Data Preprocessing: The dataset is preprocessed to remove newline characters from the articles.

Model Initialization: The PegasusForConditionalGeneration model is loaded from the "google/pegasus-large" checkpoint using the get\_hf\_objects function. This model is specifically designed for text summarization tasks.

Summarization Preprocessing: A SummarizationPreprocessor is initialized with parameters for tokenizing the input articles and target abstracts, specifying maximum token lengths, and minimum summary character length.

DataBlock and DataLoader: The data is processed into a form suitable for training using a Seq2SeqTextBlock and a custom Seq2SeqBatchTokenizeTransform. This ensures that the input articles and target abstracts are tokenized appropriately for the model.

Metrics: Various summarization metrics such as ROUGE, BERTScore, BLEU, METEOR, and SACREBLEU are specified for evaluation during training.

Model Training: A BaseModelWrapper is initialized with the pre-trained Pegasus model. A Learner object is then created with the data, model, optimizer, loss function, and callbacks for metrics calculation and model checkpointing.

Precision Handling: The learner is set to use 16-bit floating-point precision (to\_fp16) for faster training and lower memory usage.

Learning Rate Finder: The optimal learning rate is found using the learning rate finder (lr\_find), which suggests a learning rate based on the loss curve.

Fine-Tuning: The model is fine-tuned for 10 epochs using the fit\_one\_cycle method with the suggested learning rate. Metrics are calculated and displayed after each epoch.

5. Generate a 1 - 2 page note report for the patient to view:

Now we will use the AlpaCare LLM to apply it on the generated summary. This LLM will be finetuned on the current local dataset and trained to obtain relevant medications and precautionary measures to be taken by the patient.

I have built the model. I further need to do the same transformer fine tuning for the data and customarily tokenize it to make it even more customized to the use case.   
  
Package Imports: I import the AutoModelForCausalLM and AutoTokenizer classes from the Transformers library, which allows me to use pre-trained language models for text generation tasks.

Model Initialization: I specify the model name or path (model\_name\_or\_path) of the pre-trained language model I want to use, such as "xz97/AlpaCare-llama1-7b". I load the tokenizer and model using AutoTokenizer.from\_pretrained and AutoModelForCausalLM.from\_pretrained respectively. These components are then moved to the available device (GPU or CPU).

Input Preparation: I tokenize the input text (input\_text) using the loaded tokenizer, specifying padding, truncation, and maximum length. This prepares the input text for processing by the model.

Text Generation: I use the model.generate method to generate text based on the tokenized input. I specify the maximum length of the generated text (max\_length) and the number of sequences to generate (num\_return\_sequences).

6. Calendar Scheduling:

Now that we need to generate a scheduled calendar, I propose we use a federated Learning approach where the inputs from various interconnected hospital’s devices used by various affiliated entities like the health care staffs, nurses etc. will contribute to the lively and dynamically changing optimized calendar to view a schedule for the patients as well as the doctor’s free time.

In this proposed approach, we aim to leverage inputs from various devices used within our hospital, including those used by healthcare staff, nurses, and other affiliated entities, to dynamically optimize the scheduling calendar. These devices will contribute data such as patient schedules and doctor availability, providing a comprehensive view of appointments and availability within our hospital. The key idea is to use federated learning, a privacy-preserving machine learning technique, to train a model on decentralized data without sharing sensitive information.

Each device within our hospital will collect relevant data, including patient appointments and doctor's schedules, and store this information locally. This approach ensures that sensitive patient information remains within our hospital's network, maintaining privacy and security. A federated learning model, such as a neural network, will be initialized on a central server within our hospital. This model will serve as the foundation for generating the optimized calendar, learning from the collective knowledge of our hospital's data.

During the training phase, the local data from each device within our hospital will be used to update the federated learning model. However, instead of sending raw data to the central server, only model updates (i.e. weights) will be transmitted. This method ensures that sensitive patient information is not exposed during the training process, preserving privacy and security within our hospital. The central server will aggregate the model updates from all devices within our hospital to update the federated learning model, allowing it to learn from the collective knowledge of our hospital's data while maintaining data privacy.

Once the model is trained and updated, it can generate a scheduled calendar that optimizes patient appointments and doctor's free time within our hospital. The calendar will be dynamically adjusted based on new data and schedule changes, providing an accurate and up-to-date view of appointments and availability within our hospital. This approach not only improves efficiency in scheduling but also ensures the privacy and security of patient data within our hospital.

7. Dashboard Portal Application Development:

So, the plan is to create a portal dashboard for our hospital staff and doctors. This dashboard will help them manage patient information, appointments, and access relevant medical resources. Here's how we're going to do it:

Patient ID and Dashboard Creation: We'll use Flask and Django to create a secure login system. Once logged in, each doctor will have their own dashboard. Patient IDs will be randomly generated for privacy and uniqueness. Django will handle user authentication and data processing, while Flask will be used for lighter tasks.

Integration with Federated Learning for Scheduling: The dashboard will integrate with our Federated Learning algorithm for scheduling. This means doctors will only see patients who fit into their available slots. We'll use AJAX or WebSocket for real-time updates to the calendar.

Access to Patient Summaries and Medical Records: Patient summaries and medical records will be accessible on the dashboard. We'll use the Conformer model for transcriptions, redacting personal info, and encrypting with the BB84 protocol. A caching mechanism will ensure quick access.

Retrieval of Relevant Medical Resources: An AI model will analyze the patient's illness and fetch relevant medical books from our private Google Drive repository. This will be done by integrating the Google Drive API with Django, allowing the AI model to search and retrieve documents based on keywords or topics related to the patient's condition.