Documentation for Custom ChatBot:

1. Model Chosen for the Task:

   RNN (Recurrent Neural Network) and LSTM (Long Short-Term Memory) models are well-suited for building chatbots due to their ability to handle sequential data and capture temporal dependencies in conversations. Some key features of RNN and LSTM based models are:

   a. Sequential Nature: Chatbot conversations can be seen as sequences of text, where each message is dependent on the previous messages in the conversation. RNNs are designed to handle sequential data, making them a natural choice for modeling chatbot interactions.
   b. Contextual Understanding: RNNs, especially LSTM, are capable of capturing long-range dependencies in the conversation, allowing the chatbot to understand the context of the ongoing dialogue. This is crucial for generating coherent and contextually relevant responses.
   c. Transfer Learning: Pre-training an LSTM model on a large corpus of text data (e.g., using Word2Vec, GloVe, or other word embeddings) can provide a useful starting point for building a chatbot. Transfer learning can accelerate training and improve performance, especially when the chatbot has limited training data.

2. Dataset Preparation:

I used the chatterbot/english dataset which is available on Kaggle. It is a conversational dataset which comprises of several topics such as AI, Computers, Food, etc.

For Data preparation, I used tensorflow text tokenizer to convert input text into tokens. Moreover, the tokens were all lower cased during processing.

I created a vocabulary of all the unique words that were present in the dataset. The length of the vocabulary was 1894.

Additionally, I also used pre-trained word embeddings GloVe. This was done because using a pre-trained vocabulary can be beneficial, especially when you have a small dataset. It allows you to leverage the knowledge and representation power of a larger corpus.

3. Training the model:

I trained the model with the following hyperparameters:

Batch size = 50

Epochs = 150

Metric used = Accuracy

During the training procedure, the loss and accuracy were loss: 1.2434 - Accuracy: 0.0972 at epoch 1. But, at the end of 150 epochs, these values were loss: 0.0576 - accuracy: 0.9543.

4. Inference:

I saved the tokenizer, tokenizer_params, encoder_model and the decoder_model as .h5 file so that I could load these values during the time of inference.