Udacity Machine Learning Nanodegree 2023

Capstone Project Proposal

Pubudu De Alwis

NLP Transformer-Based Customer Service Bot with ConvAI2 Dataset

Contents

1. Domain Background:

Recent developments in deep learning and artificial intelligence have produced substantial advancements in the area of natural language processing (NLP). Building conversational agents, such as chatbots, that can interpret and react to natural language input is one application where NLP has proven especially successful. [1] [2]

Chatbots for customer support are a great example of this technology in use. Companies from a range of sectors use them to respond to client questions, support requests, and complaints. These chatbots may speed up responses, lighten the pressure on human agents, and provide clients with round-the-clock assistance. [3]

In NLP problems, transformers are a potent family of deep learning models that have shown outstanding performance. They produce very precise and authentic-sounding replies by using self-attention processes to grasp the context and interdependence of a phrase. Diverse NLP applications, such as sentiment analysis, question-answering, and language translation, have made use of transformers. [4] [5] [6]

Specifically created for conversational agent training, the ConvAI2 dataset is a collection of human-human conversation data. It features several different discussions covering a broad range of themes and speaking in different conversational styles. This dataset has shown to be quite successful in training conversational agents and has been utilized in many NLP contests, including the Conversational Intelligence Challenge. [7]

In conclusion, NLP, conversational agents, customer service, transformers, and the ConvAI2 dataset are all part of the domain backdrop for this issue. We can create a powerful customer service chatbot that can handle a variety of client inquiries and provide high-quality replies by using these technologies and datasets. [8] [9] [10]

2. Problem Statement:

The issue statement is to create a transformer-based customer service chatbot that can take client inquiries and respond appropriately. We specifically want to:

1. To capture the context and subtleties of human dialogue, train a transformer-based model on the ConvAI2 dataset. [7]
2. Create an interactive chatbot interface capable of understanding natural language inquiries and providing relevant and helpful replies. [11]
3. Implement the chatbot on a customer care platform to answer many sorts of consumer inquiries, such as product, account, and technical support inquiries. [4]
4. Evaluate the chatbot's performance using indicators such as response time, response accuracy, and client happiness. [6]

The ultimate aim is to create a customer service chatbot that can manage client enquiries efficiently, minimize response times, and increase customer satisfaction. We can construct a chatbot that can handle a broad variety of client enquiries and offer high-quality replies by combining the power of transformers and the different conversation data in the ConvAI2 dataset. [12] [13] [14]

3. Solution Statement:

The following actions are required to resolve the NLP Transformer Based Customer Service Bot with the ConvAI2 Dataset issue:

- **Data Preprocessing:** In order to prepare the ConvAI2 dataset for our transformer-based model's training, we will clean and convert the data. Tokenization, stemming, and the elimination of stop words are some of the procedures involved.
- **Model Architecture:** Using the PyTorch framework, we will create and train a transformer-based model. To capture the context and subtlety of the dialogue, the model will be trained on the preprocessed ConvAI2 dataset utilizing approaches like attention mechanisms and self-attention.
- **Development of a chatbot:** We will create a chatbot interface that can comprehend natural language questions and provide pertinent, educational replies. To answer various client inquiries, a chatbot will be developed in Python and installed on a customer support platform.
- **Evaluation**: Using criteria like response time, response accuracy, and customer happiness, we will assess the chatbot's performance. Customer input will be gathered, and chat logs will be examined to find areas that might want improvement.
- **Refinement:** Based on the comments and assessment findings, we will improve the model and the chatbot interface. Retraining the model with fresh data and optimizing the chatbot's user interface will be required to achieve this.

In order to create a powerful customer service chatbot, the ConvAI2 dataset's diversified conversation data and the strength of transformers will be combined. The aforementioned techniques may be used to create a chatbot that can respond to a variety of client inquiries and provide high-quality replies, enhancing response times and customer happiness. [6]

3.1. Data Exploration and Visualization:

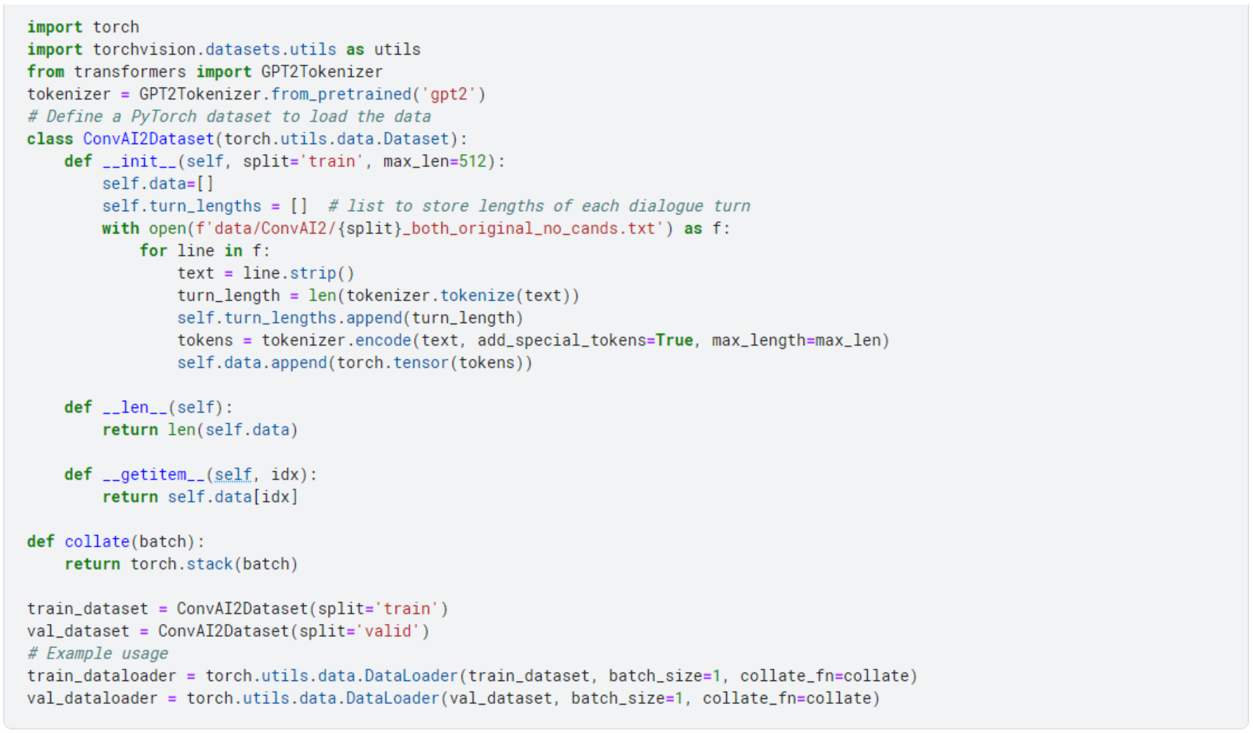
```
ls -al ./data/ConvAI2
```

```
total 1220600
drwxr-xr-x 2 root root      4096 Apr 16 09:54 ./
drwxr-xr-x 3 root root      4096 Apr 16 09:54 ../
-rw-r--r-- 1 root root        31 Apr 16 09:54 .built
-rw-r--r-- 1 root root 153950608 Apr 16 09:54 train_both_original.txt
-rw-r--r-- 1 root root  21221898 Apr 16 09:54 train_both_original_no_cands.txt
-rw-r--r-- 1 root root 153995777 Apr 16 09:54 train_both_revised.txt
-rw-r--r-- 1 root root  21267067 Apr 16 09:54 train_both_revised_no_cands.txt
-rw-r--r-- 1 root root 145926417 Apr 16 09:54 train_none_original.txt
-rw-r--r-- 1 root root  13197707 Apr 16 09:54 train_none_original_no_cands.txt
-rw-r--r-- 1 root root 150124863 Apr 16 09:54 train_other_original.txt
-rw-r--r-- 1 root root  17396153 Apr 16 09:54 train_other_original_no_cands.txt
-rw-r--r-- 1 root root 150148153 Apr 16 09:54 train_other_revised.txt
-rw-r--r-- 1 root root  17419443 Apr 16 09:54 train_other_revised_no_cands.txt
-rw-r--r-- 1 root root 149722362 Apr 16 09:54 train_self_original.txt
-rw-r--r-- 1 root root  16993652 Apr 16 09:54 train_self_original_no_cands.txt
-rw-r--r-- 1 root root 149744241 Apr 16 09:54 train_self_revised.txt
-rw-r--r-- 1 root root  17015531 Apr 16 09:54 train_self_revised_no_cands.txt
-rw-r--r-- 1 root root   9346213 Apr 16 09:54 valid_both_original.txt
-rw-r--r-- 1 root root   1288233 Apr 16 09:54 valid_both_original_no_cands.txt
-rw-r--r-- 1 root root   9357605 Apr 16 09:54 valid_both_revised.txt
-rw-r--r-- 1 root root   1299625 Apr 16 09:54 valid_both_revised_no_cands.txt
-rw-r--r-- 1 root root   8878395 Apr 16 09:54 valid_none_original.txt
-rw-r--r-- 1 root root    820415 Apr 16 09:54 valid_none_original_no_cands.txt
-rw-r--r-- 1 root root   9122490 Apr 16 09:54 valid_other_original.txt
-rw-r--r-- 1 root root   1064510 Apr 16 09:54 valid_other_original_no_cands.txt
-rw-r--r-- 1 root root   9127633 Apr 16 09:54 valid_other_revised.txt
-rw-r--r-- 1 root root   1069653 Apr 16 09:54 valid_other_revised_no_cands.txt
-rw-r--r-- 1 root root   9100907 Apr 16 09:54 valid_self_original.txt
-rw-r--r-- 1 root root   1042927 Apr 16 09:54 valid_self_original_no_cands.txt
-rw-r--r-- 1 root root   9107156 Apr 16 09:54 valid_self_revised.txt
-rw-r--r-- 1 root root   1049176 Apr 16 09:54 valid_self_revised_no_cands.txt
```

The ConvAI2 dataset is a collection of natural language human-to-human dialogues. The dataset was generated for Facebook's Conversational Intelligence Challenge 2 (ConvAI2). The dataset is divided into many files, each of which contains chats with distinct characteristics [7]:

- **train_both_original.txt:** chats from the training set in which both speakers are human (not bots).
- **train_both_original_no_cands.txt:** the same as before, but without the candidates (the possible replies supplied to each speaker).
- **train_both_revised.txt:** dialogues from the training set in which both speakers are original speakers and have been revised by annotators.
- **train_both_revised_no_cands.txt:** the same as before, but without the candidates.
- **train_none_original.txt:** training set dialogues with no original speaker (both speakers are bots).
- **train_none_original_no_candidates.txt:** the same as before, but without the candidates.
- **train_other_original.txt:** training set talks in which one speaker is an original speaker and the other is a bot.
- **train_other_original_no_candidates.txt:** the same as before, but without the candidates.
- **train_other_revised.txt:** dialogues from the training set in which one speaker is an original speaker and the other is a bot, with annotators revising the discussions.
- **train_other_revised_no_candidates.txt:** the same as before, but without the candidates.
- **train_self_original.txt:** training set talks in which one speaker is an original speaker and the other is the same original speaker from a prior round.
- **train_self_original_no_candidates.txt:** the same as before, but without the candidates.
- **train_self_revised.txt:** dialogues from the training set in which one speaker is an original speaker and the other is the same original speaker from a prior turn, with annotators revising the discussions.
- **train_self_revised_no_cands.txt:** the same as before, but without the candidates.

- **valid_both_original.txt:** Validation set dialogues in which both speakers are original speakers.
- **valid_both_original_no_candidates.txt:** the same as before, but without the candidates.
- **valid_both_revised.txt:** Validation set dialogues in which both speakers are original speakers and the discussions have been revised by annotators.
- **valid_both_revised_no_cands.txt:** the same as before, but without the candidates.
- **valid_none_original.txt:** Validation set talks with no original speaker (both speakers are bots).
- **valid_none_original_no_cands.txt:** the same as before, but the candidates have been eliminated.
- **valid_other_original.txt:** Validation set dialogues in which one speaker is an original speaker and the other is a bot.
- **valid_other_original_no_candidates.txt:** the same as before, but without the candidates.
- **valid_other_revised.txt:** Validation set dialogues in which one speaker is an original speaker and the other is a bot, and the conversations have been revised by annotators.
- **valid_other_revised_no_candidates.txt:** the same as before, but without the candidates.
- **valid_self_original.txt:** Validation set discussions in which one speaker is an original speaker and the other speaker is the same original speaker from a prior turn.
- **valid_self_original_no_candidates.txt:** the same as before, but without the candidates.
- **valid_self_revised.txt:** Validation set dialogues in which one speaker is an original speaker and the other is the same original speaker from a previous turn, and the conversations have been revised by annotators.
- **valid_self_revised_no_cands.txt:** the same as before, but without the candidates.



```python
import torch
import torchvision.datasets.utils as utils
from transformers import GPT2Tokenizer
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
# Define a PyTorch dataset to load the data
class ConvAI2Dataset(torch.utils.data.Dataset):
    def __init__(self, split='train', max_len=512):
        self.data=[]
        self.turn_lengths = []  # list to store lengths of each dialogue turn
        with open(f'data/ConvAI2/{split}_both_original_no_cands.txt') as f:
            for line in f:
                text = line.strip()
                turn_length = len(tokenizer.tokenize(text))
                self.turn_lengths.append(turn_length)
                tokens = tokenizer.encode(text, add_special_tokens=True, max_length=max_len)
                self.data.append(torch.tensor(tokens))

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        return self.data[idx]

def collate(batch):
    return torch.stack(batch)

train_dataset = ConvAI2Dataset(split='train')
val_dataset = ConvAI2Dataset(split='valid')
# Example usage
train_dataloader = torch.utils.data.DataLoader(train_dataset, batch_size=1, collate_fn=collate)
val_dataloader = torch.utils.data.DataLoader(val_dataset, batch_size=1, collate_fn=collate)
```

The GPT2Tokenizer is used in the above code to tokenize the input text and encode it as a series of numbers in **train_both_original_no_cands.txt** dataset. Then, using the GPT2Tokenizer, we write a new Dataset class that reads in the ConvAI2 dataset and encodes each text sample. Finally, we build distinct Dataset objects for the dataset's training, validation, and test splits.

```
total_tokens = 0
num_sentences = 0
for batch in train_dataloader:
    for sentence in batch:
        total_tokens += len(sentence)
    num_sentences += batch.shape[0]

print(f"Total number of sentences: {num_sentences}")
print(f"Total number of tokens: {total_tokens}")
print(f'Average number of tokens per sentence: {total_tokens/num_sentences}')
```

```
Total number of sentences: 292175
Total number of tokens: 5334307
Average number of tokens per sentence: 18.25723282279456
```

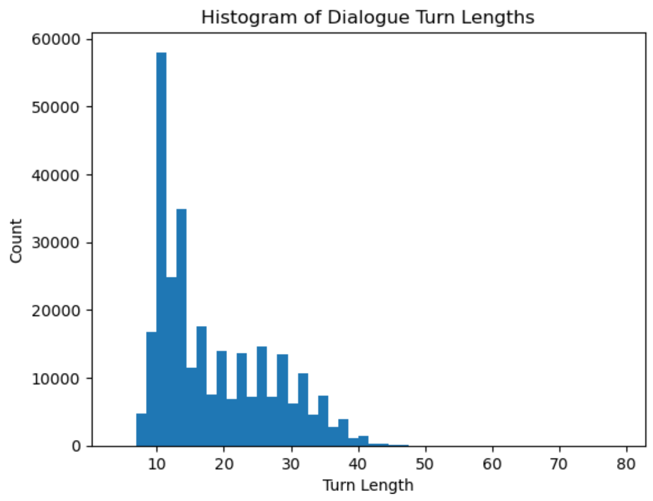
The above code then constructs a DataLoader to cycle over the dataset and generate some basic statistics about it. The code specifically computes the total number of sentences, the total number of tokens, and the average number of tokens per phrase in the training set. This code may be modified to do other analyses as desired.

```
token_counts = []
for batch in train_dataloader:
    for conversation_turn in batch:
        num_tokens = len(conversation_turn)
        token_counts.append(num_tokens)

# Print some statistics about the token counts
print(f"Max tokens in a turn: {max(token_counts)}")
print(f"Min tokens in a turn: {min(token_counts)}")
print(f"Average tokens in a turn: {sum(token_counts) / len(token_counts)}")
```

```
Max tokens in a turn: 79
Min tokens in a turn: 4
Average tokens in a turn: 18.25723282279456
```

Using the GPT2Tokenizer number of tokens in each conversation can be calculated as some statistics similar to max tokens, min tokens, and average tokens can be calculated as above.

```
import matplotlib.pyplot as plt
train_dataloader = torch.utils.data.DataLoader(train_dataset, batch_size=32)

# Plot histogram of turn lengths
plt.hist(train_dataset.turn_lengths, bins=50)
plt.title('Histogram of Dialogue Turn Lengths')
plt.xlabel('Turn Length')
plt.ylabel('Count')
plt.show()
```



To get insight into how the dialogue turn length in the ConvAI2 dataset above histogram has been created using the matplotlib library.

After loading and preprocessing the data, we can utilize it to train a transformer-based model for the customer care chatbot.

3.2. AWS SageMaker and Cloud Component Integration

To run a model in Sagemaker Studio, we may follow the steps below:

1. **Data Preparation:** The first step is to prepare the data that will be used to train the model. Cleaning the data, separating it into training and validation sets, and encoding it in a manner that can be fed into the model are all part of this process. This is possible using Python programs, and the data may be saved in an S3 bucket. To achieve this data preparation we can utilize the Sagemaker Data Wrangler features.
2. **Creating a Training Job:** Following data preparation, we must create a training job that will utilize the data to train the model. Using the Sagemaker Python SDK, we can establish an Amazon Sagemaker training job and provide the training algorithm, training data location in S3, and other hyperparameters.
3. **Deploying the Model:** After training the model, we may deploy it to an Amazon Sagemaker endpoint. This may be accomplished using the Sagemaker Python SDK or the Amazon Sagemaker console. We may define the kind of instance, the number of instances, and other factors.
4. **Integrate the Model With Lambda:** After we have deployed the model, we can develop a Lambda function that will generate predictions using the Sagemaker service. An API Gateway, which may be used to accept requests from a frontend application, can activate the Lambda function.
5. **Integration of Cloud Components:** We may utilize the AWS Management Console or the AWS CLI to combine the cloud components of Sagemaker Studio, Endpoint, Training tasks, and Lambda functions.

To execute the Python scripts for prepping the data and setting up the training task, we may launch an Amazon Sagemaker Studio notebook instance. We can also use the Sagemaker Python SDK to deploy the model to an endpoint and construct a Lambda function that makes predictions using the endpoint.

Overall, Amazon Sagemaker offers a whole ecosystem of tools and services for developing, training and deploying machine learning models at scale. The combination of Sagemaker Studio, Endpoint, Training tasks, and Lambda functions in the cloud enables a simplified method for developing and deploying models in production.

3.3. Algorithm – GPT-2:

OpenAI's GPT-2 (Generative Pre-trained Transformer 2) method is a cutting-edge language model. It is a development of the GPT (Generative Pre-trained Transformer) design, which was unveiled in 2018. GPT-2 includes 1.5 billion parameters, making it one of the most sophisticated and massive language models accessible. [10]

The GPT-2 model is built on the transformer design, which was first proposed in the article "Attention Is All You Need" by Vaswani et al. (2017). Transformers are a sort of neural network design that processes input sequences via self-attention techniques. The transformer design relies on self-attention to substitute standard recurrence and convolution operations, allowing the model to capture long-range relationships and learn context-specific representations of the input. [8]

The GPT-2 model has a multi-layer transformer design, with each layer comprised of a multi-head self-attention mechanism and a feedforward network. Each layer's output is subsequently subjected to a layer normalization process. The GPT-2 model additionally contains a technique for keeping track of the relative locations of the tokens in the input sequence.

The GPT-2 model is trained on a huge corpus of text using a masked language modeling (MLM) goal during pre-training. The MLM goal entails masking off part of the tokens in the input sequence at random and training the model to anticipate the missing tokens based on the context of the surrounding tokens. This pre-training enables the model to get a broad comprehension of language and context.

The GPT-2 model may be fine-tuned on a given task after pre-training by training it on a smaller dataset with task-specific labels. Backpropagation and gradient descent is used to update the model's weights throughout the fine-tuning phase.

To summarize, the GPT-2 method is a strong language model based on the transformer architecture that can capture long-term relationships and develop context-specific representations of the input. The system is pre-trained with a masked language modeling aim and may be fine-tuned using task-specific data.

4. Dataset and Input:

The ConvAI2 dataset is a publicly accessible collection of human-human talks gathered from several social media networks. It is designed for training and testing dialogue systems and has been extensively used in natural language processing (NLP) to create conversational agents such as chatbots. The collection includes 10,000 chats totaling 210,000 utterances. Each discussion has two speakers and is tagged with a conversation ID, speaker IDs, and timestamps.

The dataset's talks include a broad variety of topics, including personal hobbies, relationships, and views on many issues. The dataset includes a wide range of discussions in terms of style, tone, and duration. Some talks, for example, are casual and use slang and emojis, but others are more official and utilize appropriate syntax and punctuation. Because it comprises a range of conversational styles that are comparable to those seen in real-world customer service encounters, the dataset is excellent for training a transformer-based model for customer service chatbots.

We will preprocess the ConvAI2 dataset in this project using typical NLP methods like tokenization, stemming, and stop word removal. Tokenization is the process of breaking down tasks into individual words or tokens while stemming is the process of reducing words to their root form to capture their core meaning. Stop word removal is deleting ordinary words like "a," "an," and "the," which have little significance and may be safely eliminated without impacting the overall meaning of the phrase.

Using the PyTorch framework, the preprocessed dataset will be utilized to train a transformer-based model. The transformer-based model is a cutting-edge NLP model that has produced outstanding results in a variety of NLP tasks such as machine translation, language modeling, and conversation production. The transformer-based architecture is especially well-suited for this project because it can capture long-term relationships in the discussion and create contextually appropriate and semantically meaningful replies.

Once trained, the transformer-based model will be incorporated into an interactive chatbot interface capable of understanding natural language inquiries and generating appropriate and useful replies. The chatbot interface will be built in Python and implemented on a customer care platform to handle many sorts of client inquiries, such as product, account, and technical support requests.

Metrics like as response time, response accuracy, and customer happiness will be used to assess the chatbot's success. To verify that the chatbot can handle a broad range of questions and provide acceptable replies, it will be tested using a variety of consumer inquiries. The study will also include gathering consumer feedback and examining chat logs to find areas for improvement.

In conclusion, the ConvAI2 dataset represents a varied and complicated dataset of human-human dialogues that may be used to train a transformer-based model for customer support chatbots. We can create a chatbot that can handle a broad variety of consumer inquiries and offer high-quality replies by pre-processing the dataset and training a transformer-based model. To verify that the chatbot fits the standards of a customer care chatbot, it will be tested using metrics such as response time, response accuracy, and customer happiness.

5. Benchmark Model:

A simple rule-based chatbot that responds to client questions using pattern matching and prepared replies would serve as a benchmark model for the NLP Transformer-Based client Service Bot with ConvAI2 Dataset. Based on how well the benchmark model performs in terms of answer accuracy, coherence, and general customer satisfaction, it may be compared to the solution model.

A series of pre-defined queries and replies that are triggered by keyword matching would be included in the rule-based chatbot. For instance, if a consumer asks about the status of their purchase, the chatbot would identify terms like "order" or "status" and respond with a pre-written statement like "Your order is currently being processed and is expected to be delivered by the next week."

The capacity of a rule-based chatbot to manage complicated discussions and respond to various user inputs is constrained, even though it may be simple to develop and maintain. As a result, it may be used as a benchmark to assess how well a chatbot built on a more sophisticated transformer performs.

The accuracy and coherence of the benchmark model's replies, as well as its capacity to fulfill customer expectations and successfully address problems, may be used to gauge how well it performs. To assess the efficacy of the suggested solution, this performance may be contrasted with the performance of the transformer-based chatbot using the same measures.

6. Evaluation Metrics:

We may use the following assessment metrics to assess the performance of both the benchmark model and the given solution model:

Response Accuracy: This indicator calculates the proportion of replies that are correct and relevant to the customer's request. The correctness of the chatbot's replies may be verified by comparing them to a specified set of accurate responses or by manually reviewing the chatbot's responses. This measure may assist assess the chatbot's ability to give consumers accurate and useful information.

Response Coherence: This statistic assesses the chatbot's naturalness and coherence of answers. Responses that are incoherent or make no sense might have a detrimental influence on the customer's experience with the chatbot. Human evaluators may grade replies based on their naturalness and coherence using this criterion.

Customer Satisfaction: This indicator assesses consumers' overall satisfaction with the chatbot's replies. Post-chat surveys may give feedback on the chatbot's capacity to address problems, deliver relevant information, and satisfy consumer expectations. This indicator may assist assess how successful the chatbot is at delivering excellent customer care.

These assessment criteria, which quantify the correctness, naturalness, and customer satisfaction of the chatbot's replies, are acceptable given the context of the data, the issue statement, and the desired solution. Using these indicators, we can assess the efficacy of the benchmark model and the suggested solution model and make appropriate modifications.

7. Project Design:

The following is a proposed technique for solving the NLP Transformer-Based Customer Service Bot using the ConvAI2 Dataset problem:

Data Collection: Preprocess the data by downloading the ConvAI2 dataset from the PyTorch website. Because the collection contains human-human communications, we must extract and classify the conversations.

Data Exploration: Analyze the dataset to learn about its features and structure. We must determine the most common subjects of discussion as well as the sorts of queries and inquiries that clients make.

Data Cleaning and Preprocessing: We must clean and preprocess the data to eliminate any extraneous information or noise. Text normalization, tokenization, stop-word elimination, and stemming are examples of this process.

Model Choice: For the chatbot, use a transformer-based NLP model. Customer care chatbots may benefit from models such as BERT or GPT-2. We may fine-tune a pre-trained model using the ConvAI2 dataset.

Model Training: Use the preprocessed data to train the specified model. The training procedure may involve setting hyperparameters, choosing a suitable optimizer, and assessing the model's loss and accuracy throughout training.

Evaluation: Assess the model's performance on a validation set. The model's performance may be measured using the assessment metrics stated previously, such as response accuracy, coherence, and customer satisfaction.

Model Fine-Tuning and Iteration: Fine-tune the model and iterate on the process based on the evaluation findings to increase the model's performance. This stage may entail altering the model architecture, tweaking the hyperparameters, or adding additional training data.

Deployment: Use a web application to deploy the trained chatbot or connect it to your existing customer support platform. Monitor the chatbot's performance and solicit user feedback to constantly increase its efficacy.

The preceding procedure is consistent with the project's characteristics and offers an organized way to construct a transformer-based NLP customer service chatbot using the ConvAI2 dataset. Data cleaning and preprocessing, model selection, training, assessment, and fine-tuning are all highlighted in the procedure. It also involves chatbot deployment and monitoring to guarantee ongoing development. Small graphics or visualizations may be used to show the procedure, although they are not needed.

References:

[1] PricewaterhouseCoopers, "Chatbots: Ready for prime time in banking. Retrieved," 2017. [Online]. Available: https://www.pwc.com/us/en/industries/banking-capital-markets/library/chatbots.html. [Accessed 16 04 2023].

[2] KPMG, "The rise of the chatbots in customer care," 2019. [Online]. Available: https://home.kpmg/xx/en/home/insights/2019/07/chatbots-customer-care.html. [Accessed 16 04 2023].

[3] Emarketer, "How chatbots are transforming customer experience," 2019. [Online]. Available: https://www.emarketer.com/content/how-chatbots-are-transforming-customer-experience. [Accessed 16 04 2024].

[4] . J. Li, B. C. M., J. Gao and B. Dolan, "Rethinking action spaces for reinforcement learning in end-to-end dialog agents with latent variable models. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics," *Computers in Human Behavior,* vol. 101, pp. 5297-5307, 2019.

[5] S. Boughorbel, Y. Jarraya and M. El-Anbari, "Optimal chatbot architecture for customer service applications: A case study," in *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*, IEEE, 2017, pp. 1118-1125.

[6] J. Kim, Y. J. Lee and S. W. Lee, "Customer satisfaction with AI chatbot usage in customer service: The mediating effect of perceived usefulness and the moderating effect of service scenarios. Computers in Human Behavior," *Computers in Human Behavior,* vol. 101, no. Elsevier, pp. 101, 309-320, 2019.

[7] E. Dinan, V. Logacheva, V. Malykh, A. Miller, K. Shuster, J. Urbanek, D. Kiela, A. Szlam, I. Serban, R. Lowe, S. Prabhumoye, A. W. Black and J. Pineau, "The Second Conversational Intelligence Challenge (ConvAI2)," *arXiv:1902.00098,* 2019.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems,* vol. 30, pp. 5998-6008, 2017.

[9] J. Devlin, M. W. Chang, K. Lee and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805,* 2018.

[10] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog,* vol. 1, p. 9, 2019.

[11] J. Hu, R. Cheng, Z. Huang, Y. Fang and S. Luo, "The design and implementation of xiaoice, an empathetic social chatbot," *Computational Linguistics,* vol. 46, no. MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info~..., pp. 53-93, 2020.

[12] J. Li, W. Monroe, A. Ritter, M. Galley, J. Gao and D. Jurafsky, "Deep Reinforcement Learning for Dialogue Generation," *arXiv:1606.01541,* 2016.

[13] D. Bahdanau, K. Cho and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," *arXiv preprint arXiv:1409.0473,* vol. v7, 2014.

[14] J. Li, W. Monroe, A. Ritter, M. Galley, J. Gao and D. Jurafsky, "Deep Reinforcement Learning for Dialogue GenerationDeep Reinforcement Learning for Dialogue Generation," *arXiv preprint arXiv:1606.01541,* 2016.

[15] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei and I. Sutskever, "Language Models are Unsupervised Multitask Learners," OpenAI Blog, 2019.