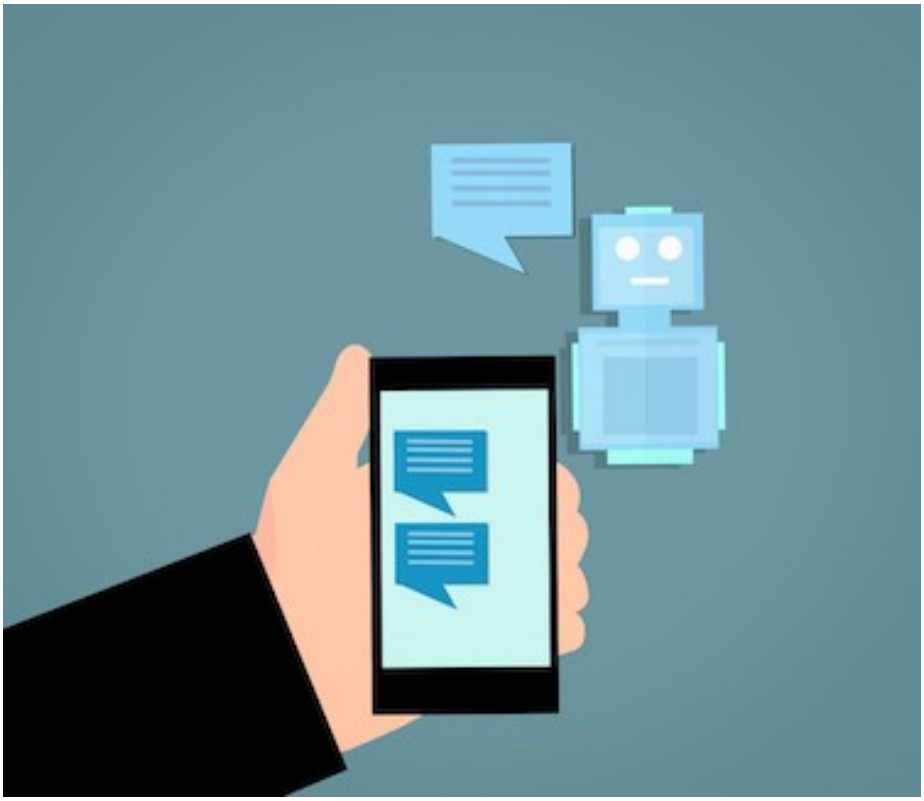
Create Simple Chatbot with Open Source LLMs using Python and Hugging Face

In this lab, you will create a very simple but functional chatbot!



Learning outcomes:

At the end of this lab, you will be able to:

- Describe the main components of a chatbot
- Explain what an LLM is
- Select an LLM for your application
- Describe how a transformer essentially works
- Feed input into a transformer (tokenization)
- Program your own simple chatbot in Python

Introduction: Under the hood of a chatbot

Intro: How does a chatbot work?

A chatbot is a computer program that takes a text input, and returns a corresponding text output.

Chatbots use a special kind of computer program called a transformer, which is like its brain. Inside this brain, there is something called a language model (LLM), which helps the chatbot understand and generate human-like responses. It deciphers many examples of human conversations it has seen prior to responding in a sensible manner.

Transformers and LLMs work together within a chatbot to enable conversation. Here's a simplified explanation of how they interact:

- **Input processing:** When you send a message to the chatbot, the transformer helps process your input. It breaks down your message into smaller parts and represents them in a way that the chatbot can understand. Each part is called a token.
- **Understanding context:** The transformer passes these tokens to the LLM, which is a language model trained on lots of text data. The LLM has learned patterns and meanings from this data, so it tries to understand the context of your message based on what it has learned.
- **Generating response:** Once the LLM understands your message, it generates a response based on its understanding. The transformer then takes this response and converts it into a format that can be easily sent back to you.
- **Iterative conversation:** As the conversation continues, this process repeats. The transformer and LLM work together to process each new input message, understand the context, and generate a relevant response.

The key is that the LLM learns from a large amount of text data to understand language patterns and generate meaningful responses. The transformer helps with the technical aspects of processing and representing the input/output data, allowing the LLM to focus on understanding and generating language.

Once the chatbot understands your message, it uses the language model to generate a response that it thinks will be helpful or interesting to you. The response is sent back to you, and the process continues as you have a back-and-forth conversation with the chatbot.

Intro: Hugging Face

Hugging Face is an organization that focuses on natural language processing (NLP) and AI. They provide a variety of tools, resources, and services to support NLP tasks.

You'll be making use of their Python library `transformers`in this project.

Alright! Now that you know how a chatbot works at a high level, let's get started with implementing a simple chatbot!

Step 1: Installing requirements

Follow these steps to create a Python virtual environment and install the necessary libraries. Open a new terminal first.
Set up your virtual environment:

```
pip3 install virtualenv
virtualenv my_env # create a virtual environment my_env
source my_env/bin/activate # activate my_env
```

For this example, you will be using the `transformers` library, which is an open-source natural language processing (NLP) toolkit with many useful features, and also let's install a torch library.

```
python3 -m pip install transformers==4.30.2 torch
```

Wait a few minutes to install the packages.

To create a new Python file, Click on `File Explorer`, then right-click in the explorer area and select `New File`. Name this new file `chatbot.py`.

Open **chatbot.py** in IDE

Import required tools from the transformers library, choose and fetch the model

Step 2: Import our required tools from the transformers library

For this example, you will be using `AutoTokenizer` and `AutoModelForSeq2SeqLM` from the `transformers` library.

```
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
```

Add the step into the `chatbot.py` Python file.

Step 3: Choosing a model

Choosing the right model for your purposes is an important part of building chatbots! You can read on the different types of models available on the Hugging Face website: https://huggingface.co/models.

LLMs differ from each other in how they are trained. Let's look at some examples to see how different models fit better in various contexts.

- **Text generation**:
  If you need a general-purpose text generation model, consider using the GPT-2 or GPT-3 models. They are known for their impressive language generation capabilities.
  Example: You want to build a chatbot that generates creative and coherent responses to user input.

- **Sentiment analysis**:
  For sentiment analysis tasks, models like BERT or RoBERTa are popular choices. They are trained to understand the sentiment and emotional tone of text.
  Example: You want to analyze customer feedback and determine whether it is positive or negative.

- **Named entity recognition**:
  LLMs such as BERT, GPT-2, or RoBERTa can be used for Named Entity Recognition (NER) tasks. They perform well in understanding and extracting entities like person names, locations, organizations, etc.
  Example: You want to build a system that extracts names of people and places from a given text.

- **Question answering**:
  Models like BERT, GPT-2, or XLNet can be effective for question-answering tasks. They can comprehend questions and provide accurate answers based on the given context.
  Example: You want to build a chatbot that can answer factual questions from a given set of documents.

- **Language translation**:
  For language translation tasks, you can consider models like MarianMT or T5. They are designed specifically for translating text between different languages.
  Example: You want to build a language translation tool that translates English text to French.

However, these examples are very limited and the fit of an LLM may depend on many factors such as data availability, performance requirements, resource constraints, and domain-specific considerations. It's important to explore different LLMs thoroughly and experiment with them to find the best match for your specific application.

Other important purposes that should be taken into consideration when choosing an LLM include (but are not limited to):

- Licensing: Ensure you are allowed to use your chosen model the way you intend
- Model size: Larger models may be more accurate, but might also come at the cost of greater resource requirements
- Training data: Ensure that the model's training data aligns with the domain or context you intend to use the LLM for
- Performance and accuracy: Consider factors like accuracy, runtime, or any other metrics that are important for your specific use case

To explore all the different options, check out the available models on the Hugging Face website.

For this example, you'll be using `facebook/blenderbot-400M-distill` because it has an open-source license and runs relatively fast.

```
model_name = "facebook/blenderbot-400M-distill"
```

Add this step into your `chatbot.py` python file.

Step 4: Fetch the model and initialize a tokenizer

When running this code for the first time, the host machine will download the model from Hugging Face API. However, after running the code once, the script will not re-download the model and will instead reference the local installation.

You'll be looking at two terms here: `model` and `tokenizer`.

In this script, you initiate variables using two handy classes from the `transformers` library:

- `model` is an instance of the class `AutoModelForSeq2SeqLM`, which allows you to interact with your chosen language model.
- `tokenizer` is an instance of the class `AutoTokenizer`, which optimizes your input and passes it to the language model efficiently. It does so by converting your text input to “tokens”, which is how the model interprets the text.

```
# Load model (download on first run and reference local installation for consequent runs)
model = AutoModelForSeq2SeqLM.from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)
```

Conversation details

Step 5: Chat

Now that you're all set up, let's start chatting!

There are several things you'll do to have an effective conversation with your chatbot.

Before interacting with your model, you need to initialize an object where you can store your conversation history.

1. Initialize object to store conversation history

Afterward, you'll do the following for each interaction with the model:

2. Encode conversation history as a string
3. Fetch prompt from user
4. Tokenize (optimize) prompt
5. Generate output from the model using prompt and history
6. Decode output
7. Update conversation history

Step 5.1: Keeping track of conversation history

The conversation history is important when interacting with a chatbot because the chatbot will also reference the previous conversations when generating output.

For your simple implementation in Python, you may use a list. Per the Hugging Face implementation, you will use this list to store the conversation history as follows:

```
conversation_history
```

[input_1, output_1, input_2, output_2, …]

Let's initialize this list before any conversations occur.

```
conversation_history = []
```

Add this step to your python code.

Step 5.2: Encoding the conversation history

During each interaction, you will pass your conversation history to the model along with your input so that it may also reference the previous conversation when generating the next answer.

The `transformers` library function you are using expects to receive the conversation history as a string, with each element separated by the newline character `'\n'`. Thus, you create such a string.

You'll use the `join()` method in Python to do exactly that. (Initially, your history_string will be an empty string, which is okay, and will grow as the conversation goes on).

```
history_string = "\n".join(conversation_history)
```

Add this to `chatbot.py`

Step 5.3: Fetch prompt from user

Befor you start building a simple terminal chatbot, let's look at an example of the input:

```
input_text ="hello, how are you doing?"
```

Add this to `chatbot.py`

Step 5.4: Tokenization of user prompt and chat history

Tokens in NLP are individual units or elements that text or sentences are divided into. Tokenization or vectorization is the process of converting tokens into numerical representations. In NLP tasks, you often use the `encode_plus` method from the `tokenizer` object to perform tokenization and vectorization. Let's encode your inputs (prompt & chat history) as tokens so that you may pass them to the model.

```
inputs = tokenizer.encode_plus(history_string, input_text, return_tensors="pt")
print(inputs)
```

Add this to `chatbot.py` and run it:

```
python3 chatbot.py
```

In doing so, you've now created a Python `dictionary` which contains special keywords that allow the model to properly reference its contents.

To learn more about tokens and their associated pretrained vocabulary files, you can explore the pretrained_vocab_files_map attribute. This attribute provides a mapping of pretrained models to their corresponding vocabulary files.

```
tokenizer.pretrained_vocab_files_map
```

Add this to `chatbot.py`

Step 5.5: Generate output from the model

Now that you have your inputs ready, both past and present inputs, you can pass them to the model and generate a response. According to the documentation, you can use the `generate()` function and pass the inputs as keyword arguments (kwargs).

```
outputs = model.generate(**inputs)
print(outputs)
```

Add this to `chatbot.py` and run it:

```
python3 chatbot.py
```

Great - now you have your outputs! However, the current output `outputs` is also a dictionary and contains tokens, not words in plaintext.

Therefore, you just need to decode the first index of `outputs` to see the response in plaintext.

> Please note that the model used in this project is a basic, lightweight version, not intended for handling complex queries. For more advanced and robust LLMs, you can explore a wide range of options at huggingface.com.

Step 5.6: Decode output

You may decode the output using `tokenizer.decode()`. This is known as "detokenization" or "reconstruction". It is the process of combining or merging individual tokens back into their original form, to reconstruct the original text or sentence.

```
response = tokenizer.decode(outputs[0], skip_special_tokens=True).strip()
print(response)
```

Add this to `chatbot.py` and run it:

```
python3 chatbot.py
```

The output:

Alright! You've successfully had an interaction with your chatbot! You've given it a prompt, and received its response.

Now, all that's left to do is to update your conversation history, so that you may pass it with the next iteration.

Step 5.7: Update conversation history

All you need to do here is add both the input and response to `conversation_history` in plaintext.

```
conversation_history.append(input_text)
conversation_history.append(response)
print(conversation_history)
```

Add this to `chatbot.py` and run it:

```
python3 chatbot.py
```

Step 6: Repeat

You have gone through all the steps of interacting with your chatbot. Now, you can put everything in a loop and run a whole conversation!

```
while True:
    # Create conversation history string
    history_string = "\n".join(conversation_history)
```

```
        # Get the input data from the user
        input_text = input("> ")
        # Tokenize the input text and history
        inputs = tokenizer.encode_plus(history_string, input_text, return_tensors="pt")
        # Generate the response from the model
        outputs = model.generate(**inputs)
        # Decode the response
        response = tokenizer.decode(outputs[0], skip_special_tokens=True).strip()

        print(response)
        # Add interaction to conversation history
        conversation_history.append(input_text)
        conversation_history.append(response)
```

Add this to `chatbot.py` and run it:

```
python3 chatbot.py
```

The output:

Voila! YOu have built a simple, functional chatbot that you can interact with through your terminal!

Press `cntrl + c` to exit the conversation.

Conclusion

Congratulations, you have successfully completed this lab!
This lab showcases the creation of a basic chatbot using Python and Hugging Face's open-source language models. The project guides learners through the key concepts of chatbots, including understanding and using transformers and tokenization. It provides a step-by-step approach to setting up the environment, choosing a model, and coding a chatbot capable of holding a conversation. This hands-on project, crafted by Dr. Sina Nazeri, is a great resource for beginners

interested in exploring AI and chatbot development, demonstrating how to build a functional chatbot in an accessible and engaging way.

Author

Sina Nazeri (Ph.D.)	**linkedin**
	As a data scientist in IBM, I have always been passionate about sharing my knowledge and helping others learn about the field. I believe that everyone should have the opportunity to learn about data science, regardless of their background or experience level. This belief has inspired me to become a learning content provider, creating and sharing educational materials that are accessible and engaging for everyone.