Set Up a Simple Gradio Interface to Interact with Your Models

Estimated time: 30 minutes

Imagine you're developing a customer support chatbot for a company. The goal is to provide a seamless and interactive way for customers to get answers to their questions. To achieve this goal, you need an interface where users can input their queries and receive responses generated by a large language model (LLM). This lab guides you through the creation of such an interface using Gradio. You'll learn how to integrate various Gradio components, including text input fields, buttons, and display elements, to create an intuitive and user-friendly customer support experience.

Creating an interface is crucial for several reasons:

- **User accessibility**: A front-end interface makes it easy for users to interact with the LLM without needing technical expertise.
- **Enhanced user experience**: An intuitive interface can provide a better user experience, making interactions smoother and more efficient.
- **Customization**: Gradio allows you to customize the interface to meet specific needs, whether for a chatbot, a data analysis tool, or other applications.
- **Seamless integration**: Gradio's flexibility enables you to seamlessly integrate the front-end with various backend technologies, including LLMs.

Source: DALL-E

In this lab, you will learn how to use Gradio to set up a front-end interface that allows users to interact with a backend LLM. Gradio is a powerful and user-friendly library for creating customizable web-based interfaces. By the end of this lab, you will have the skills to use essential and commonly used Gradio elements to build an interface that communicates with an LLM, enabling you to create a functional chatbot.

This lab will provide you with hands-on experience in building an interactive interface, empowering you to create applications that leverage the power of LLMs through user-friendly front-end solutions.

Learning objectives

By the end of this project, you will be able to:

- **Use Gradio to build interactive front-end interfaces**, enabling users to interact with backend LLMs seamlessly
- **Create a functional chatbot**, allowing users to input queries and receive responses from an LLM
- **Implement essential and commonly used Gradio elements**, such as text input fields, buttons, and display areas, to enhance the users' experience
- **Customize and deploy web-based applications**, facilitating various use cases including customer support, data analysis, and others

Set up your virtual environment

Set up a virtual environment

Let's create a virtual environment. Using a virtual environment allows you to manage dependencies for different projects separately, avoiding conflicts between package versions.

In the terminal of your Cloud IDE, verify that you are in the path `/home/project`, then run the following commands to create a Python virtual environment.

Note: A terminal window should be open by default in the lower part your Cloud IDE environment. If it is not, go to `Terminal` –> `New Terminal` in the menu bar at the top in order to open a new terminal window.

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

Install the necessary libraries

To ensure that your script successfully runs, you first need to consider that for certain functions within these scripts rely on external libraries, it's essential to install some prerequisite libraries before you begin.

For this project, the you will need key libraries including `Gradio` for creating user-friendly web interfaces, `IBM Watsonx AI` for leveraging the advanced LLM model from IBM watsonx's API and LangChain.

- `gradio` allows you to build interactive web applications quickly, making your AI models accessible to users with ease.
- `ibm-watsonx-ai` for using LLMs from IBM's watsonx.ai.
- `langchain`, `langchain-ibm`, `langchain-community` for using relevant features from LangChain.

Next, still using your terminal connection, install the following packages:

```
# installing necessary pacakges in my_env
python3.11 -m pip install \
gradio==4.44.0 \
pydantic==2.10.6 \
```

```
ibm-watsonx-ai==1.1.2 \
langchain==0.2.11 \
langchain-community==0.2.10 \
langchain-ibm==0.1.11
```

Now the environment is ready to create Python files.

A quick tutorial of Gradio

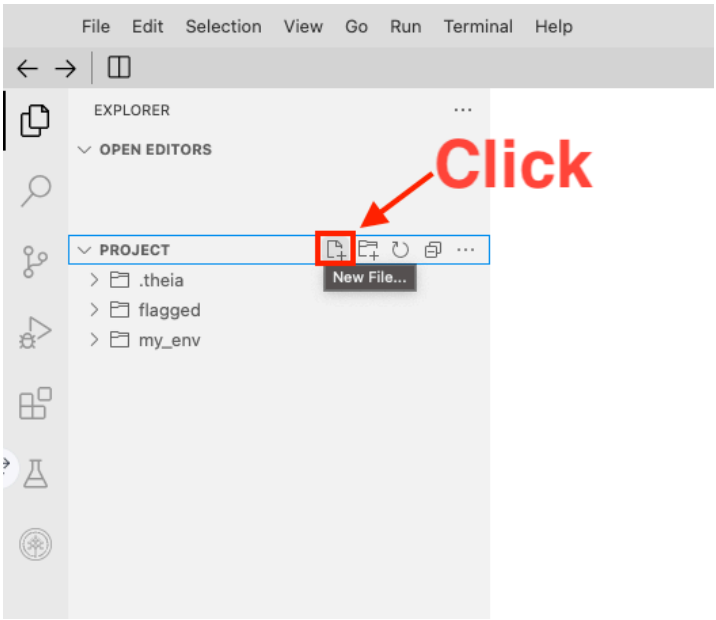
Gradio overview

You can use Gradio to create a web interface that seamlessly bridges the gap between machine learning models and end-user interactions. Gradio enables developers to effortlessly transform their algorithms into accessible, interactive applications. With Gradio, you can quickly design web interfaces where users can input data in various format including text, images, or audio, and instantly view the output generated by your model. For more information, please visit Gradio.

Create a Gradio demo

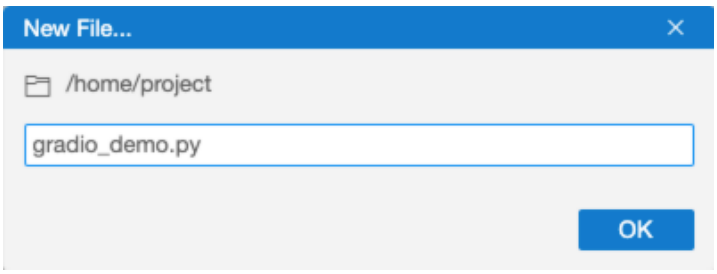
Create a .py file

First, you need to create a `gradio_demo.py` file in cloud IDE. To accomplish this task, navigate to the the **EXPLORER** icon in the left pane, select the **PROJECT** drop-down arrow, and select the **New file** icon on the menu bar as seen in the following image.



Alternatively, for some browser types, you might need to right-click **PROJECT** and select **New file**.

Whichever method you used to create your file, name the file`gradio_demo.py` and select **OK**.



Next explore demo that explains how you can create a sum calculator that uses Gradio.

Create a demo sum calculator

You will create an application that can calculate the sum of your input numbers.

Note that the `gr.Number()` element from Gradio creates a numeric field for the user to enter numbers as input or can display the numeric output.

In the `gradio_demo.py`, you can enter the following code:

```
import gradio as gr
def add_numbers(Num1, Num2):
    return Num1 + Num2
# Define the interface
demo = gr.Interface(
    fn=add_numbers,
    inputs=[gr.Number(), gr.Number()], # Create two numerical input fields where users can enter numbers
    outputs=gr.Number() # Create numerical output fields
)
# Launch the interface
demo.launch(server_name="127.0.0.1", server_port= 7860)
```

Launch the demo application

Return to the terminal window and verify that the virtual environment `my_env` label appears at the start of the line. This verificaton confirms that you are in the `my_env` environment that you just created. Next, use the following command to run the Python script.

```
python3.11 gradio_demo.py
```

After this code runs successfully, you will see a message in the terminal window with the local URL displayed.You'll also see a message that instructs you that if you want to create a public link, to set `share=True` in `Launch()`.

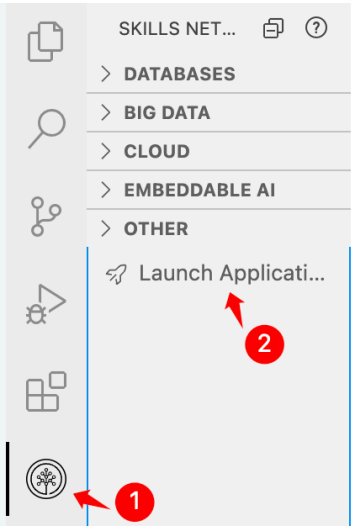
```
Running on local URL:  http://127.0.0.1:7860

To create a public link, set `share=True` in `launch()`.
```

Because the web application is hosted locally on port 7860, select the following **Web Application** button to view the application you developed.

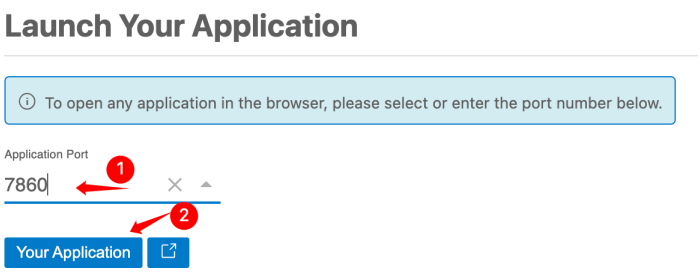
Web Application

(Note: if this **Web Application** button does not work, follow these pictured instructions to launch the app.)

1. Select **OTHER** and **Launch Application**.



2. Next, type your port number in the **Application Port** field and select **Your Application**



Now let's take a look at the web application, which shows an example of sum of 3 and 4. You're encouraged to experiment with the web app's inputs and outputs!

If you want to stop the script, you can press `Ctrl+C` in the terminal and close the appliction window.

Exercise

Can you create a Gradio application that can combine two input sentences together? Use what you know from the demo and your Python knowledge to create this app. Take your time to complete this exercise.

▶ Let's take a look of the answer.

You just had a first experience with the Gradio interface! It's easy, right? If you want to learn more about customization in Gradio, you can try out this guided project **Bring your Machine Learning model to life with Gradio**. You can find more relevant courses and projects on cognitiveclass.ai

For the rest of this project, you will use Gradio as an interface for the created application.

Gradio Interface Inputs and Outputs

Common Inputs

Gradio has a large number of input types. The more commonly encountered ones are listed below:

- `Checkbox`: A checkbox that can be set to `True` or `False`.
- `CheckboxGroup`: An input type that allows users to select multiple values from a predefined checkbox list.
- `Dropdown`: An input type that provides a dropdown list where, by default, one value can be selected. If `multiselect` is set to `True`, then one or more values can be selected.
- `File`: An input type that allows a user to upload a file.
- `Image`: An input type that allows the user to select or upload an image.
- `Radio`: An input type that forces the user to choose one value.
- `Slider`: An input type that provides a slider where a value must be selected between a minimum and a maximum range. The `value` parameter defines the default value, and `step` provides the increment value. Setting the minimum, maximum, and `step` values to integers will select integer values.
- `Textbox`: An expandible text box that allows the user to type in text.

Common Outputs

Many of Gradio's input types can also function as outputs. The available output types depend on the output of the function provided to `Interface`. In practice, for most LLM applications, the output type is typically text. As such, a suitable choice is either `gr.Textbox()`, or just `"text"`, which offers an expandable text box.

Another frequently encountered output type is `Label`. `Label` is typically used for classification tasks, and can output the predicted probabilities of each class. If you have a large number of classes, you can use the `num_top_classes` parameter to control the number of classes that are outputted. For instance, if you have 1000 classes, setting `Label(num_top_classes=3)` would output just the three classes with the highest predicted probabilities instead of the predicted probabilities for all classes.

An Example of Common Input Types

Let's have a look at an example of some of Gradio's common input types.

First, create a file called `common_input_types.py` and paste the following script into it:

```
import gradio as gr
def sentence_builder(quantity, tech_worker_type, countries, place, activity_list, morning):
    return f"""The {quantity} {tech_worker_type}s from {" and ".join(countries)} went to the {place} where they {" and ".join(activity_list)} until the {"morning
demo = gr.Interface(
    fn=sentence_builder,
    inputs=[
        gr.Slider(3, 20, value=4, step=1, label="Count", info="Choose between 3 and 20"),
        gr.Dropdown(
            ["Data Scientist", "Software Developer", "Software Engineer"],
            label="tech_worker_type",
            info="Will add more tech worker types later!"
        ),
        gr.CheckboxGroup(["Canada", "Japan", "France"], label="Countries", info="Where are they from?"),
        gr.Radio(["office", "restaurant", "meeting room"], label="Location", info="Where did they go?"),
        gr.Dropdown(
            ["partied", "brainstormed", "coded", "fixed bugs"],
            value=["brainstormed", "fixed bugs"],
            multiselect=True,
            label="Activities",
            info="Which activities did they perform?"
```

```
            ),
            gr.Checkbox(label="Morning", info="Did they do it in the morning?"),
        ],
        outputs="text",
        examples=[
            [3, "Software Developer", ["Canada", "Japan"], "restaurant", ["coded", "fixed bugs"], True],
            [4, "Data Scientist", ["Japan"], "office", ["brainstormed", "partied"], False],
            [10, "Software Engineer", ["Canada", "France"], "meeting room", ["brainstormed"], False],
            [8, "Data Scientist", ["France"], "restaurant", ["coded"], True],
        ]
    )
    demo.launch(server_name="127.0.0.1", server_port= 7860)
```

After saving `common_input_types.py`, run the script by pasting the following into the terminal window and clicking `Enter`:

```
python3.11 common_input_types.py
```

Finally, launch the app by clicking the button below, or by using the workaround described on the previous page:

Web Application

In addition to illustrating the different types of inputs, the above also produces a table with input examples. These examples are generated by configuring the `examples` parameter in `Interface` as a list of lists, where each inner list represents one example, or a single row in the examples table.

When you are done interacting with the app, hit `Ctrl+C` in the terminal, and close the application window.

IBM watsonx.ai LLM quickstart

watsonx.ai introduction

Watsonx.ai is IBM's commercial generative AI and scientific data platform based on cloud technology that includes a studio, data store, and governance toolkit designed to support multiple LLMs. This platform is designed to support a diverse array of AI development tasks, providing developers with access to IBM’s Granite models, Meta’s Llama models, Mistral AI’s Mixtral model, and various others.

In this section, you will learn how to use watsonx.ai's API to create a simple Q&A bot. This bot will leverage the advanced capabilities of watsonx.ai to understand and respond to user queries accurately. Whether you're new to programming or an experienced developer, this step-by-step tutorial will equip you with the knowledge to integrate watsonx.ai's LLMs into your applications.

Create a Q&A bot

This tutorial will walk you through the process of creating a Q&A chatbot using either the Granite 3.3 8B or the Mixtral 8x7B model. You’ll also learn how to select the model as you work your way through the tutorial. These powerful foundation models are seamlessly integrated into IBM's watsonx.ai platform, simplifying your development journey. The provided API eliminates the need for generating complex API tokens, streamlining your application creation process. The Granite 3.3 8B and Mixtral 8x7B models both support the following use cases:

- Questions & Answers (Q&A)
- Summarization
- Classification
- Generation
- Extraction
- Retrieval-augmented generation
- Code generation

Moreover, the Granite 3.3 8B model supports these additional use cases:

- Reasoning & planning
- Fill-in-the-middle

Since you are creating a Q&A chatbot, the Granite 3.3 8B and the Mixtral 8x7B models align perfectly for this task. For a description of available models on watsonx.ai, please refer here.

Note: The Granite 3.3 8B and Mixtral 8x7B models, like any AI technology, have their limitations, and it is possible to encounter nonsensical responses occasionally. The primary objective of this project is to provide guidence for how to use LLMs with watsonx.ai.

Follow these step-by-step instructions to create your application:

1. Still in the **PROJECT** directory, create a new Python file named `simple_llm.py`. (You are welcome to choose a different name if you prefer).

2. Enter the following script content into your newly created `simple_llm.py` file and save your changes. Explanations of the various parts of the code are provided by in-code comments.

```
# Import the necessary packages
from ibm_watsonx_ai.foundation_models import ModelInference
from ibm_watsonx_ai.metanames import GenTextParamsMetaNames as GenParams
from ibm_watsonx_ai import Credentials
from langchain_ibm import WatsonxLLM
# Specify the model and project settings
# (make sure the model you wish to use is commented out, and other models are commented)
#model_id = 'mistralai/mixtral-8x7b-instruct-v01' # Specify the Mixtral 8x7B model
model_id = 'ibm/granite-3-3-8b-instruct' # Specify IBM's Granite 3.3 8B model
# Set the necessary parameters
parameters = {
    GenParams.MAX_NEW_TOKENS: 256,  # Specify the max tokens you want to generate
    GenParams.TEMPERATURE: 0.5, # This randomness or creativity of the model's responses
}
project_id = "skills-network"
# Wrap up the model into WatsonxLLM inference
watsonx_llm = WatsonxLLM(
    model_id=model_id,
    url="https://us-south.ml.cloud.ibm.com",
    project_id=project_id,
    params=parameters,
)
# Get the query from the user input
query = input("Please enter your query: ")
# Print the generated response
print(watsonx_llm.invoke(query))
```

3. Open your terminal and verify that you are operating within the virtual environment (`my_env`) you previously established.

4. Run the following command in the terminal.

```
python3.11 simple_llm.py
```

After the code successfully runs, you can input the following query in the terminal. The app then generates a response.

The following image displays an example text response based on the query: `How to be a good data scientist?`.

In the code, you used `"skills-network"` as `project_id` to gain immediate, free access to the API without the need for initial registration.

Important! This access method is exclusive to this cloud IDE environment. If you are interested in using the model/API in a local environment, detailed instructions and further information are available in this tutorial.

Moreover, note that, by default, this app uses IBM's Granite 3.3 8B model. In order to use Mistral AI's Mixtral 8x7B model instead, all you have to do is change the `model_id` parameter. The necessary instructions for doing so are embedded in the script> Moreover, the code on the following page will provide you with a specific example of building a Q&A bot with Mixtral 8x7B.

Gradio and the LLM

Integrate the application into Gradio

Having successfully created a Q&A bot with your script, you might notice that responses are only displayed in the terminal. You might wonder if it's possible to integrate this application with Gradio so that you can provide a web interface for inputting questions and receiving responses.

The following code guides you through this integration process. It includes the following three components:

- Initializing the model
- Defining the function that generates responses from the LLM
- Constructing the Gradio interface and enabling interaction with the LLM
    - The `gr.Textbox` element is being used to create text field and hold users' input query and LLM's output

```
# Import necessary packages
from ibm_watsonx_ai.foundation_models import ModelInference
from ibm_watsonx_ai.metanames import GenTextParamsMetaNames as GenParams
from ibm_watsonx_ai import Credentials
from langchain_ibm import WatsonxLLM
import gradio as gr
# Model and project settings
#model_id = 'mistralai/mixtral-8x7b-instruct-v01' # Specify the Mixtral 8x7B model
model_id = 'ibm/granite-3-3-8b-instruct' # Specify IBM's Granite 3.3 8B model
# Set necessary parameters
```

```
parameters = {
    GenParams.MAX_NEW_TOKENS: 256,  # Specify the max tokens you want to generate
    GenParams.TEMPERATURE: 0.5, # This randomness or creativity of the model's responses
}
project_id = "skills-network"
# Wrap up the model into WatsonxLLM inference
watsonx_llm = WatsonxLLM(
    model_id=model_id,
    url="https://us-south.ml.cloud.ibm.com",
    project_id=project_id,
    params=parameters,
)
# Function to generate a response from the model
def generate_response(prompt_txt):
    generated_response = watsonx_llm.invoke(prompt_txt)
    return generated_response
# Create Gradio interface
chat_application = gr.Interface(
    fn=generate_response,
    allow_flagging="never",
    inputs=gr.Textbox(label="Input", lines=2, placeholder="Type your question here..."),
    outputs=gr.Textbox(label="Output"),
    title="Watsonx.ai Chatbot",
    description="Ask any question and the chatbot will try to answer."
)
# Launch the app
chat_application.launch(server_name="127.0.0.1", server_port= 7860)
```

Next, perform the following steps:

1. Navigate to the `PROJECT` directory, right-click and create a new file named `llm_chat.py`.
2. Input the code provided above into this new llm_chat.py file.
3. Open your terminal and verify that you are in the `my_env` virtual environment.
4. Run the following code in the terminal to run the application.

```
python3.11 llm_chat.py
```

After the code runs successfully, you will see message that tells you which local URL the application is using. You'll also see a message that instructs you that if you want to create a public link, to set `share=True` in `Launch()`.

```
Running on local URL:  http://127.0.0.1:7860

To create a public link, set `share=True` in `launch()`.
```

5. Select the following **Web Application** button to launch and view the application.

Web Application

The chatbot you successfully created is displayed, appearing as follows:

Now, you can ask yout chatbot a question.

Here is an example of a question asked of the chatbot and its displayed output.

When you are ready to terminate the script, press `Ctrl+C` in the terminal and close the application window.

Finally, note that this application used the Mixtral 8x7B model by default. Please compare it to the `simple_llm.py` script that used the Granite 3.3 8B model instead. With watsonx.ai all we had to do was change `model_id` to change the model!

Exercise

You might observe that responses from the LLM are occasionally incomplete. How can you identify the cause of this issue? Also, can you modify the code to enable the model to generate more content? You can!

Actually, all you need to do is:

▶ Click here for the answer

Authors

Authors

Kang Wang
Ricky Shi
Wojciech "Victor" Fulmyk

Other contributors

Joseph Santarcangelo