Your Personalized Job Application Coach Based on LLM

Estimated time needed: 60 minutes

Project overview

Embark on a journey to master the integration of cutting-edge large language model (LLM) into user-friendly applications.

You will start by gaining hands-on experience with Gradio, creating a simple yet powerful sample application. As you dive deeper, you will unlock the potential of watsonx.ai's large language model, learning how to harness its power for real-world applications. By the end of this project, you will be adept at developing three specialized tools: an AI-driven resume enhancer, a personalized cover letter generator, and a strategic job hunting advisor.

Each module is carefully designed to not only teach you the technical know-how but also to provide insights into the ever-evolving job market, making you well-equipped to cater to the needs of modern job seekers. "AI Career Coach" is more than just a learning experience; it's a stepping stone to a future where AI bridges the gap between talent and opportunity.

Source: DALL-E

Learning objectives

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

- **Understand and use Gradio**: Gain proficiency in using Gradio to build and deploy AI-based web applications.
- **Integrate watsonx.ai's language model**: Learn to integrate and leverage the capabilities of a large language model in application development.
- **Develop a resume enhancement tool**: Acquire the skills to create an application that uses AI to analyze and improve resumes based on job descriptions.
- **Create a personalized cover letter generator**: Master the development of an AI application that drafts customized cover letters, enhancing job application processes.
- **Build a career advice application**: Develop an innovative tool that offers personalized job hunting and career improvement advice based on AI analysis of resumes and job descriptions.
- **Apply practical AI solutions**: Understand how to apply AI in practical, real-world scenarios, particularly in the context of job applications and career development.

Setting 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, ensure that you are in the path `/home/project`, then run the following commands to create a Python virtual environment.

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

Installing necessary libraries

To ensure a seamless execution of our scripts, and considering that certain functions within these scripts rely on external libraries, it's essential to install some prerequisite libraries before we begin. For this project, the key libraries we'll need are `Gradio` for creating user-friendly web interfaces and `IBM Watson Machine Learning` for leveraging advanced LLM model.

- **Gradio package**: Gradio allows us to build interactive web applications quickly, making our AI models accessible to users with ease.
- **IBM Watson Machine Learning package**: IBM Watson Machine Learning package integrates powerful IBM LLM models into our project.

Here's how to install these packages (still from your terminal):

```
# installing necessary pacakges in my_env
python3.11 -m pip install gradio==5.12.0
python3.11 -m pip install ibm_watsonx_ai==1.1.20
python3.11 -m pip install email-validator==2.1.1
python3.11 -m pip install numpy==1.26.4
python3.11 -m pip install pandas==2.1.4
```

Now, the environment is ready to create Python files.

Quick tutorial to Gradio

Gradio overview

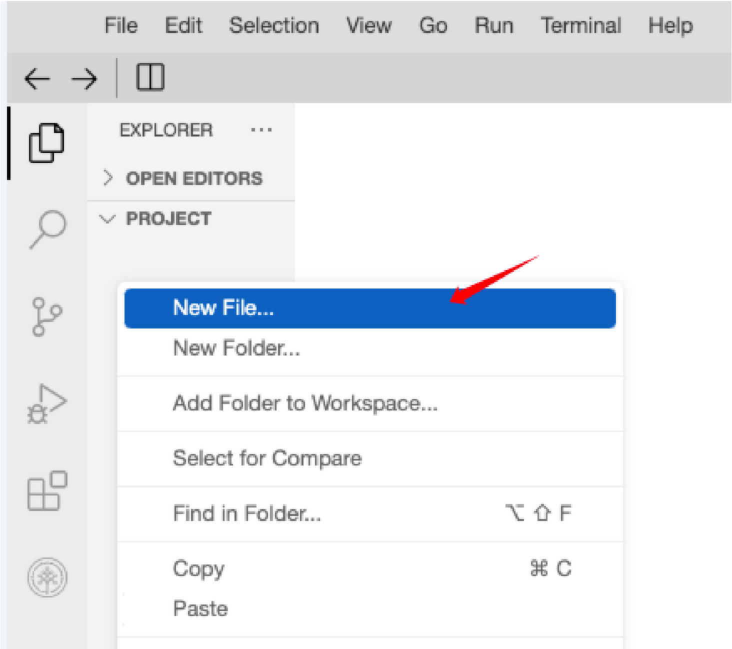
Gradio can be used to create a web interface to seamlessly bridge the gap between machine learning models and end-user interaction. It enables developers to effortlessly transform their algorithms into accessible, interactive applications. With Gradio, you can quickly design web interfaces that allow users to input data in various forms—such as text, images, or audio—and instantly view the output generated by your model. For more information, please visit Gradio.

In this project, our LLM-based resume advisor application will need a web interface to interact with. So, let's get familiar with Gradio and create a simple Gradio application first.

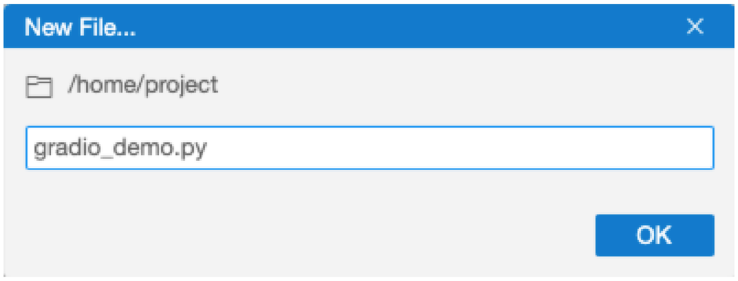
Create a Gradio demo

Create a .py file

First, we need to create a .py file in Cloud IDE on your left. Under the `EXPLORER` tab, open the `PROJECT` directory, right click and choose `New file`.



Then, you can name your file `gradio_demo.py`.



gradio_demo.py

A demo of sum calculator

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

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

```
import gradio as gr
```

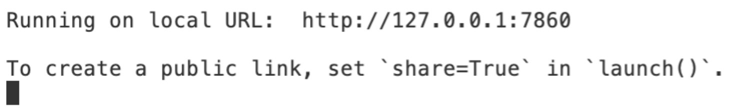
```
def add_numbers(Num1, Num2):
    return Num1 + Num2
# Define the interface
demo = gr.Interface(
    fn=add_numbers,
    inputs=["number", "number"], # Create two numerical input fields where users can enter numbers
    outputs="number" # Create numerical output fields
)
# Launch the interface
demo.launch(server_name="0.0.0.0", server_port= 7860)
```

Launch the application

Return to the terminal and verify that the virtual environment `my_env` label appears at the start of the line, which means that you are in the `my_env` environment that you just created. Next, execute the following command to run the Python script.

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

After it runs successfully, you will see a message similar to following in the terminal:



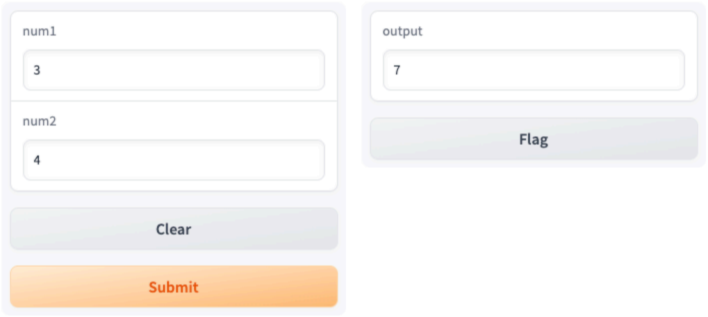
```
Running on local URL:  http://127.0.0.1:7860

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

Since the web application is hosted locally on port 7860, click on the following button to view the application we've developed.

Web Application

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



num1: 3

num2: 4

output: 7

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 which can combine two input sentences together? Take your time to complete this exercise.

▶ Let's take a look of the answer.

You just had a first taste of the Gradio interface, it\’s easy right? If you want to learn more about customization in Gradio, you are invited to take the guided project called **Bring your Machine Learning model to life with Gradio**. You can find it under Courses & Projects on cognitiveclass.ai

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

Quick start on watsonx.ai LLM

watsonx.ai introduction

watsonx.ai is IBM's commercial generative AI and scientific data platform based on cloud technology. It encompasses a studio, data store, and governance toolkit, designed to support multiple large language models (LLMs). This platform is tailored for a wide range of AI development tasks, offering developers access to IBM's own series of LLMs as well as models from Facebook's LLaMA-3 and the Hugging Face community.

In this section, we will guide you through the process of using 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 leveraging the `llama-3-2-11b-vision-instruct` model developed by Meta. This powerful foundation model has been seamlessly integrated into IBM's watsonx.ai platform, simplifying your development journey. Our provided API eliminates the need for generating complex API tokens, streamlining the application creation process. The `llama-3-2-11b-vision-instruct` model is equipped with features designed to

- Supports Q&A
- Summarization
- Classification
- Generation
- Extraction
- Retrieval-Augmented Generation tasks

This model aligns perfectly with our objective for the application being created.

Note: The `llama-3-2-11b-vision-instruct` model, like any AI technology, has its limitations, and it is possible to encounter nonsensical responses occasionally. The primary objective of this project is to provide guidence on utilizing 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. Line-by-line explanation of the code snippet is provided.

```
# Import necessary packages
from ibm_watsonx_ai import Credentials
from ibm_watsonx_ai import APIClient
from ibm_watsonx_ai.foundation_models import Model, ModelInference
from ibm_watsonx_ai.foundation_models.schema import TextChatParameters
from ibm_watsonx_ai.metanames import GenTextParamsMetaNames
# Model and project settings
model_id = "meta-llama/llama-3-2-11b-vision-instruct"  # Directly specifying the LLAMA3 model
# Set credentials to use the model
credentials = Credentials(
                   url = "https://us-south.ml.cloud.ibm.com",
                  )
# Set necessary parameters
params = TextChatParameters()
# Specifying project_id as provided
project_id = "skills-network"
# Initialize the model
model = ModelInference(
    model_id=model_id,
    credentials=credentials,
    project_id=project_id,
    params=params
)
prompt_txt = "How to be a good Data Scientist?"  # Your question
messages = [
    {
        "role": "user",
        "content": [
            {
                "type": "text",
                "text": prompt_txt
            },
        ]
    }
]
# Attempt to generate a response using the model with overridden parameters
generated_response = model.chat(messages=messages)
```

```
generated_text = generated_response['choices'][0]['message']['content']
# Print the generated response
print(generated_text)
```

3. Open your terminal and ensure 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
```

Upon successful execution of the code, a response is displayed in the terminal. An example response follows:

```
'''
To be a good data scientist, you'll need to possess a combination of technical, business, and soft skills. Here are some key areas to focus on:
**Technical Skills:**
1. **Programming languages:** Proficiency in languages such as Python, R, SQL, and Julia.
2. **Data manipulation and analysis:** Familiarity with data structures, statistical modeling, and machine learning algorithms.
3. **Data visualization:** Ability to create interactive and informative visualizations using tools like Tableau, Power BI, or D3.js.
4. **Big data technologies:** Knowledge of Hadoop, Spark, and NoSQL databases.
5. **Deep learning:** Familiarity with frameworks like TensorFlow, PyTorch, or Keras.
**Mathematical and Statistical Skills:**
1. **Linear Algebra:** Understanding of vector spaces, linear transformations, and matrix operations.
2. **Calculus:** Familiarity with derivatives, integrals, and optimization techniques.
3. **Statistics:** Knowledge of probability distributions, statistical inference, and hypothesis testing.
4. **Machine learning theory:** Understanding of the theoretical foundations of machine learning.
**Business Skills:**
1. **Communication:** Ability to effectively communicate complex technical concepts to non-technical stakeholders.
2. **Project management:** Experience with project planning, execution, and delivery.
3. **Business acumen:** Understanding of business objectives, industries, and markets.
4. **Stakeholder management:** Ability to collaborate with cross-functional teams and stakeholders.
**Soft Skills:**
1. **Problem-solving:** Ability to approach complex problems in a methodical and creative way.
2. **Critical thinking:** Capacity to analyze data, identify patterns, and draw meaningful conclusions.
3. **Collaboration:** Willingness to work with others, share knowledge, and learn from others.
4. **Adaptability:** Flexibility in adapting to new tools, technologies, and methodologies.
5. **Continuous learning:** Commitment to staying up-to-date with industry trends, technologies, and best practices.
**Additional Tips:**
1. **Practice data science:** Apply your skills to real-world problems and projects.
2. **Read industry blogs and books:** Stay informed about the latest developments in data science and AI.
3. **Network with professionals:** Attend conferences, meetups, and join online communities to connect with other data scientists.
4. **Participate in hackathons and competitions:** Showcase your skills and learn from others.
5. **Pursue certifications:** Consider obtaining certifications like Certified Data Scientist (CDS) or Certified Analytics Professional (CAP) to demo
By focusing on these areas, you'll be well on your way to becoming a proficient and effective data scientist.
'''
```

In the code, we simply used `"skills-network"` as `project_id` to gain immediate, free access to the API without the need for initial registration. It's important to note that 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.

Integrate the application into Gradio

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

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

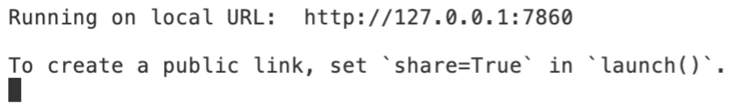
- Initializing the model.
- Defining the function that generates responses from the LLM.
- Constructing the Gradio interface, enabling interaction with the LLM.

```
# Import necessary packages
from ibm_watsonx_ai import Credentials
from ibm_watsonx_ai import APIClient
from ibm_watsonx_ai.foundation_models import Model, ModelInference
from ibm_watsonx_ai.foundation_models.schema import TextChatParameters
from ibm_watsonx_ai.metanames import GenTextParamsMetaNames
import gradio as gr
# Set credentials to use the model
credentials = Credentials(
                   url = "https://us-south.ml.cloud.ibm.com",
                  )
# Model and project settings
model_id = "meta-llama/llama-3-2-11b-vision-instruct"  # Directly specifying the LLAMA3 model
project_id = "skills-network"  # Specifying project_id as provided
params = TextChatParameters(
    temperature=0.1,
    max_tokens=256
)
# Initialize the model
model = ModelInference(
    model_id=model_id,
    credentials=credentials,
    project_id=project_id,
    params=params
)
# Function to generate a response from the model
def generate_response(prompt_txt):
    messages = [
    {
        "role": "user",
        "content": [
            {
                "type": "text",
                "text": prompt_txt
            },
        ]
    }
]
    generated_response = model.chat(messages=messages)
    generated_text = generated_response['choices'][0]['message']['content']
    return generated_text
# Create Gradio interface
chat_application = gr.Interface(
    fn=generate_response,
    flagging_mode="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()
```

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

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

After it has excuted successfully, you will see message similar to the following in the terminal:



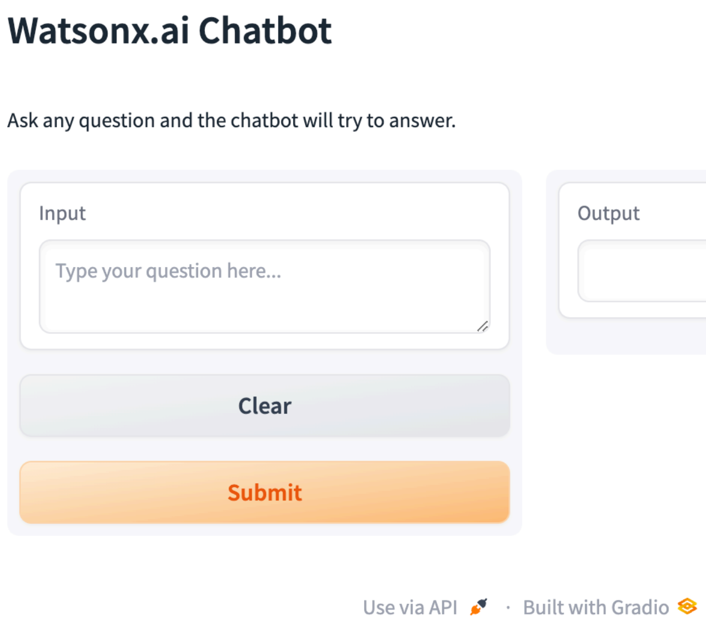
```
Running on local URL:  http://127.0.0.1:7860

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

5. Click the following button to launch and view the application.

Web Application

The chatbot you have successfully created will be displayed, appearing as follows:

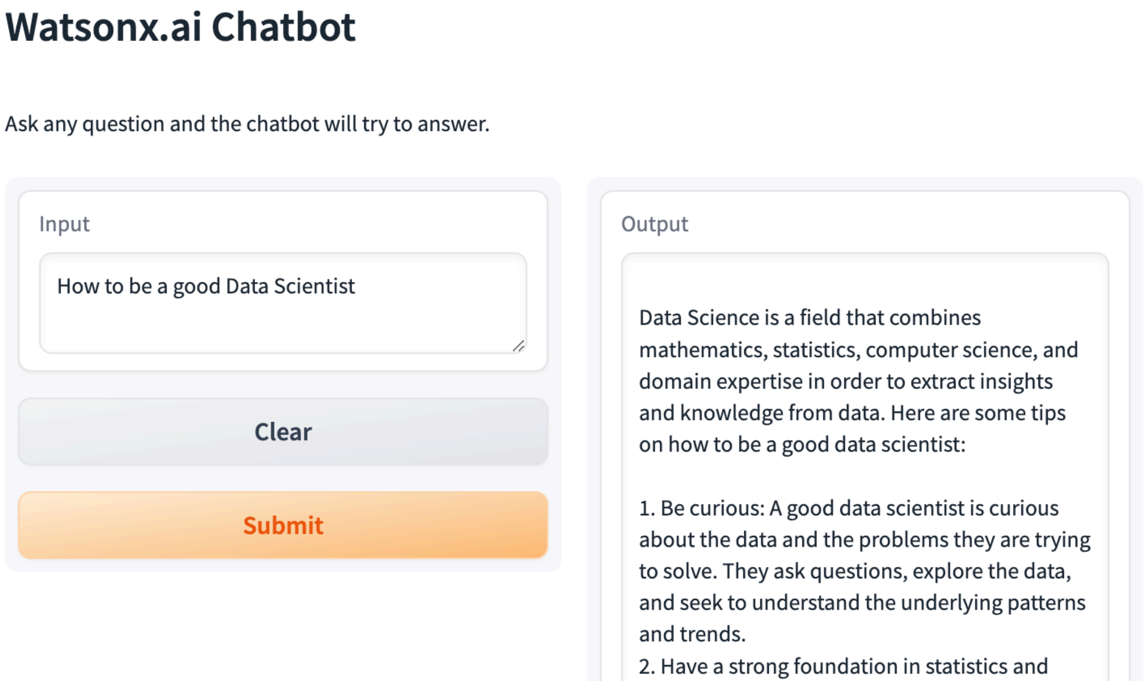


Watsonx.ai Chatbot

Ask any question and the chatbot will try to answer.

Now, feel free to ask any question to the chatbot.

Here is an example of the question asked:



Watsonx.ai Chatbot

Ask any question and the chatbot will try to answer.

How to be a good Data Scientist

```
Data Science is a field that combines
mathematics, statistics, computer science, and
domain expertise in order to extract insights
and knowledge from data. Here are some tips
on how to be a good data scientist:

1. Be curious: A good data scientist is curious
about the data and the problems they are trying
to solve. They ask questions, explore the data,
and seek to understand the underlying patterns
and trends.
2. Have a strong foundation in statistics and
```

(To terminate the script, press `Ctrl+C` in the terminal and close the appliction window.)

Exercise

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

Actually, all you need to do is:

▶ Click here for the answer

In the following section, we will explore how to develop a resume polisher using the knowledge we have just acquired.

Resume polisher

Welcome to our Resume polisher creation guide! In this section, we will guide you to create an LLM-based application tool that can help you to make your resume the best it can be. Think of your resume as your first impression for a job. You want it to look sharp, right? Our Resume polisher does just that. It checks your resume, suggests improvements, and makes sure it shines. It's like having a friend who's really good at resumes take a look at yours. Whether you need to fix up the wording, highlight your best achievements, or match your skills to what employers are looking for, we've got you covered.

Let's get started and turn your resume into something that gets you noticed and gets you the job.

1. Create a new file and name it `resume_polisher.py`.
2. Input the following code into this file.

```
# Import necessary packages
from ibm_watsonx_ai import Credentials
from ibm_watsonx_ai import APIClient
from ibm_watsonx_ai.foundation_models import Model, ModelInference
from ibm_watsonx_ai.foundation_models.schema import TextChatParameters
from ibm_watsonx_ai.metanames import GenTextParamsMetaNames
import gradio as gr
# Model and project settings
model_id = "meta-llama/llama-3-2-11b-vision-instruct"  # Directly specifying the LLAMA3 model
# Set credentials to use the model
credentials = Credentials(
                   url = "https://us-south.ml.cloud.ibm.com",
                  )
# Generation parameters
params = TextChatParameters(
    temperature=0.7,
    max_tokens=512
)
project_id = "skills-network"  # Specifying project_id as provided
# Initialize the model
model = ModelInference(
    model_id=model_id,
    credentials=credentials,
    project_id=project_id,
    params=params
)
# Function to polish the resume using the model, making polish_prompt optional
def polish_resume(position_name, resume_content, polish_prompt=""):
    # Check if polish_prompt is provided and adjust the combined_prompt accordingly
    if polish_prompt and polish_prompt.strip():
        prompt_use = f"Given the resume content: '{resume_content}', polish it based on the following instructions: {polish_prompt} for the {position
    else:
        prompt_use = f"Suggest improvements for the following resume content: '{resume_content}' to better align with the requirements and expectatio

    messages = [
    {
        "role": "user",
        "content": [
            {
                "type": "text",
                "text": prompt_use
            },
        ]
    }
]
    # Generate a response using the model with parameters
    generated_response = model.chat(messages=messages)
    # Extract and return the generated text
    generated_text = generated_response['choices'][0]['message']['content']

    return generated_text
# Create Gradio interface for the resume polish application, marking polish_prompt as optional
resume_polish_application = gr.Interface(
    fn=polish_resume,
    flagging_mode="never", # Deactivate the flag function in gradio as it is not needed.
    inputs=[
        gr.Textbox(label="Position Name", placeholder="Enter the name of the position..."),
        gr.Textbox(label="Resume Content", placeholder="Paste your resume content here...", lines=20),
        gr.Textbox(label="Polish Instruction (Optional)", placeholder="Enter specific instructions or areas for improvement (optional)...", lines=2),
    ],
    outputs=gr.Textbox(label="Polished Content"),
    title="Resume Polish Application",
    description="This application helps you polish your resume. Enter the position your want to apply, your resume content, and specific instructions
)
# Launch the application
resume_polish_application.launch()
```

3. Execute the script by entering the following command in the terminal:

```
python3.11 resume_polisher.py
```

4. Launch the application by clicking the following button.

Web Application

If the application launches successfully, it should appear as follows:

You are now ready to use this application to enhance your resume. For optimal results, we suggest polishing your resume section by section, allowing the model to perform more effectively.

Example resume content for your use.

```
'''
Designed and implemented a machine learning system that
predicts hardware malfunction with more than 80% accuracy.
'''
```

Example polishing instructions for your use.

```
'''
I use random forest model. So you should make it more specific and eye catching.
Also you should mention results it brings: cost savings by 20%.
'''
```

(To terminate the script, press `Ctrl+C` in the terminal and close the appliction window.)

Exercise

The prompt we've provided might not fully meet your needs or preferences. Feel free to modify it in the code as you see fit.

▶ Click here for the answer

Customized cover letter

In today's competitive job market, standing out from the crowd is crucial, and a personalized cover letter can be the key to catching the eye of your potential employer. Recognizing this, we introduce you to the creation of a Customized Cover Letter Generator in this section. This innovative tool is designed to help you write cover letters that not only highlight your unique skills and experiences but also tailor them to the specific job and the company you’re applying for. By inputting intended company, position, job description, and your resume, the generator will provide you with a cover letter that is both professional and personalized, ensuring that your application makes a memorable impression. Let's dive into how to create such a tool.

Following the approach of the previous code, we will now reconfigure the Gradio interface and develop a function specifically for generating the cover letter, with particular emphasis on the prompt construction. The complete code snippet follows:

```
# Import necessary packages
from ibm_watsonx_ai import Credentials
from ibm_watsonx_ai import APIClient
from ibm_watsonx_ai.foundation_models import Model, ModelInference
from ibm_watsonx_ai.foundation_models.schema import TextChatParameters
from ibm_watsonx_ai.metanames import GenTextParamsMetaNames
import gradio as gr
# Model and project settings
model_id = "meta-llama/llama-3-2-11b-vision-instruct"  # Directly specifying the LLAMA3 model
# Set credentials to use the model
credentials = Credentials(
                   url = "https://us-south.ml.cloud.ibm.com",
                  )
# Generation parameters
params = TextChatParameters(
    temperature=0.7,
    max_tokens=512
)
project_id = "skills-network"
# Initialize the model
model = ModelInference(
    model_id=model_id,
    credentials=credentials,
    project_id=project_id,
    params=params
)
# Function to generate a customized cover letter
def generate_cover_letter(company_name, position_name, job_description, resume_content):
    # Craft the prompt for the model to generate a cover letter
    prompt = f"Generate a customized cover letter using the company name: {company_name}, the position applied for: {position_name}, and the job desc
    messages = [
    {
        "role": "user",
        "content": [
            {
                "type": "text",
                "text": prompt
            },
        ]
    }
]
    # Generate a response using the model with parameters
    generated_response = model.chat(messages=messages)
    # Extract and return the generated text
    cover_letter = generated_response['choices'][0]['message']['content']
    return cover_letter
# Create Gradio interface for the cover letter generation application
cover_letter_app = gr.Interface(
    fn=generate_cover_letter,
    flagging_mode="never", # Deactivate the flag function in gradio as it is not needed.
    inputs=[
        gr.Textbox(label="Company Name", placeholder="Enter the name of the company..."),
        gr.Textbox(label="Position Name", placeholder="Enter the name of the position..."),
        gr.Textbox(label="Job Description Information", placeholder="Paste the job description here...", lines=10),
        gr.Textbox(label="Resume Content", placeholder="Paste your resume content here...", lines=10),
    ],
    outputs=gr.Textbox(label="Customized Cover Letter"),
    title="Customized Cover Letter Generator",
    description="Generate a customized cover letter by entering the company name, position name, job description and your resume."
)
# Launch the application
cover_letter_app.launch()
```

To run the code:

1. Create a new python file named `cover_letter.py`.
2. Input the code provided above.

3. In the terminal, run

```
python3.11 cover_letter.py
```

4. Launch the application by clicking the following button.

Web Application

If the application launches successfully, it should appear as follows:

You are now ready to test this application by entering the required information as prompted.

(To terminate the script, press `Ctrl+C` in the terminal and close the appliction window.)

Exercise

If you prefer to input specific technical skills keywords rather than the entire job description, could you modify the code to replace the Job Description Information input with a Job Skills Keywords input?

▶ Click here for the answer

Career advisor

In this part, we're going to create a cool tool based on LLMs that helps with your career questions. All you need to do is put in the job you're interested in, what the job is about, and your resume. Our smart tool will look at all that and give you some solid advice on how to move forward in your career. It's like having a chat with a career coach who knows all about jobs and how your skills fit in.

Similarly, we will need to redesign the Gradio interface and make corresponding adjustments to the prompt. The finalized code follows:

```
# Import necessary packages
from ibm_watsonx_ai import Credentials
from ibm_watsonx_ai import APIClient
from ibm_watsonx_ai.foundation_models import Model, ModelInference
from ibm_watsonx_ai.foundation_models.schema import TextChatParameters
from ibm_watsonx_ai.metanames import GenTextParamsMetaNames
import gradio as gr
# Model and project settings
model_id = "meta-llama/llama-3-2-11b-vision-instruct"  # Directly specifying the LLAMA3 model
# Set credentials to use the model
credentials = Credentials(
                   url = "https://us-south.ml.cloud.ibm.com",
                  )
# Generation parameters
params = TextChatParameters(
    temperature=0.7,
    max_tokens=1024
)
project_id = "skills-network"
# Initialize the model
model = ModelInference(
    model_id=model_id,
    credentials=credentials,
    project_id=project_id,
    params=params
)
# Function to generate career advice
```

```
def generate_career_advice(position_applied, job_description, resume_content):
    # The prompt for the model
    prompt = f"Considering the job description: {job_description}, and the resume provided: {resume_content}, identify areas for enhancement in the r
    messages = [
    {
        "role": "user",
        "content": [
            {
                "type": "text",
                "text": prompt
            },
        ]
    }
]

    # Generate a response using the model with parameters
    generated_response = model.chat(messages=messages)

    # Extract and format the generated text
    advice = generated_response['choices'][0]['message']['content']
    return advice
# Create Gradio interface for the career advice application
career_advice_app = gr.Interface(
    fn=generate_career_advice,
    flagging_mode="never", # Deactivate the flag function in gradio as it is not needed.
    inputs=[
        gr.Textbox(label="Position Applied For", placeholder="Enter the position you are applying for..."),
        gr.Textbox(label="Job Description Information", placeholder="Paste the job description here...", lines=10),
        gr.Textbox(label="Your Resume Content", placeholder="Paste your resume content here...", lines=10),
    ],
    outputs=gr.Textbox(label="Advice"),
    title="Career Advisor",
    description="Enter the position you're applying for, paste the job description, and your resume content to get advice on what to improve for gett
)
# Launch the application
career_advice_app.launch()
```

To run it:

1. Create a new file and name it `career_advisor.py`.
2. Enter the code provided above into this file.
3. In the terminal, run

```
python3.11 career_advisor.py
```

4. Launch the application by clicking the following button.

Web Application

If the application launches successfully, it should appear similar to:

Now play with it! 🎉

(To terminate the script, press `Ctrl+C` in the terminal and close the appliction window.)

Exercise

Change the parameter `temperature`, which controls randomness of response. The higher the temperature, the more randomness. Compare the responses between different parameter settings.

▶ Click here for the answer

Great job! 🎉 Now you have learned how to create LLM-based applications using watsonx.ai!

In the next section, we'll guide you how to use watsonx.ai locally, allowing you to run it in your own environment.

Use LLMs with watsonx.ai Locally

The `ibm_watsonx_ai` Python library allows you to work with IBM watsonx Machine Learning services, including LLMs. You can train, store, and deploy your models. You can also evaluate them using APIs. Additionally, it offers access to pre-trained, state-of-the-art LLMs for utilization via APIs, seamlessly integrating them into your application development process. Here is an introduction documentation.

The package can be installed with pip:

```
pip install ibm-watson-machine-learning
# you can also specifiy the package version such as 1.1.20
```

For example, the following is a code snippet for creating a simple QA chat with Llama2 using watsonx.ai's machine learning library.

```
from ibm_watsonx_ai import Credentials
from ibm_watsonx_ai import APIClient
from ibm_watsonx_ai.foundation_models import Model, ModelInference
from ibm_watsonx_ai.foundation_models.schema import TextChatParameters
# Set up the API key and project ID for IBM Watson
watsonx_API = "" # below is the instruction how to get them
project_id= "" # like "0blahblah-000-9999-blah-99bla0hblah0"
# Generation parameters
params = TextChatParameters(
    temperature=0.7,
    max_tokens=1024
)
model = ModelInference(
    model_id='meta-llama/llama-3-2-11b-vision-instruct',
    params=params,
    credentials={
        "apikey": watsonx_API,
        "url": "https://us-south.ml.cloud.ibm.com"
    },
    project_id=project_id
    )
q = "How to be happy?"
messages = [
        {
            "role": "user",
            "content": [
                {
                    "type": "text",
                    "text": q
                },
            ]
        }
    ]
generated_response = model.chat(messages=messages)
print(generated_response['choices'][0]['message']['content'])
```

As you might have noticed, you need to have `watsonx_API` and `project_id` for authentication when using the watsonx API. The following instructions will guide you how to get them.

Get your Own API key and project ID

watsonx API

You can get the `watsonx_API` key by first signing up for IBM Cloud at IBM watsonx account.

After you sign up and sign in to IBM watsonx account, you can follow this demonstration to create/get your IBM Cloud user API key at https://cloud.ibm.com/iam/apikeys.

Project ID

Next, you need to create/get a project ID for your watsonx.ai project. Go into your project on watsonx.ai and create a project:

In the management console, find the `project_id`.

You can see the project id in:
`Management` –> `General`
(⚠ Wait! ⚠ it is NOT done yet!). You also need to add a service to your project to make it work.

To add a service to your project (as the image below shows), after you create a project:

⇨ Go to the project's `Manage` tab.
⇨ Select the `Service and integrations` page.
⇨ Click `Associate Services`.
⇨ Add the `Watson Machine Learning` service to it.

🎉 Done! 🎉

Now, you can input the `watsonx_API` and `project_id` into the code snippet and try it. The sample output should be similar to this:

Author(s)

Hailey Quach is a Data Scientist at IBM .

Ricky Shi is a Data Scientist at IBM .

Changelog

Date	Version	Changed by	Change Description
2025/02/05	1.1	Hailey Quach	Updated lab