Lab Guide: Getting Started with Generative AI in watsonx.ai

Author: Elena Lowery, elowery@us.ibm.com

Contents

Overview

In this lab you will learn how to implement generative AI use cases in **watsonx.ai**. *Watsonx.ai* is an AI platform which can be used to implement both traditional machine learning use cases and use cases that utilize *Large Language Models* (LLMs).

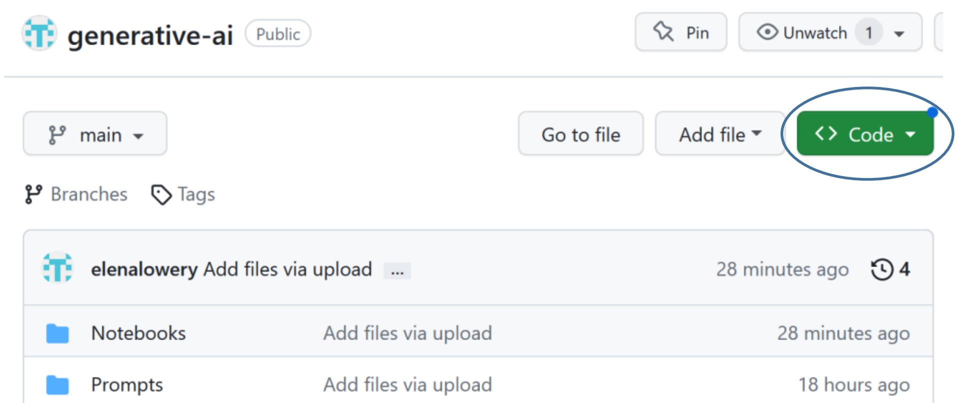
We will take a closer look at the following use cases:

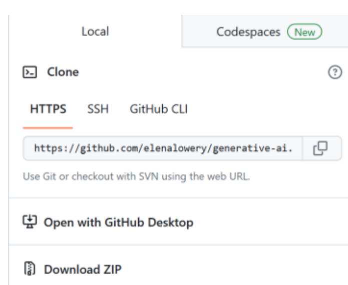
- Generation
- Summarization
- Classification

Note: LLMs are a type of a foundation model. In IBM tools and documentation, the terms LLM and foundation models are used interchangeably.

Required software, access, and files

- To complete this lab, you will need access to *watsonx.ai*. Your instructor will provide the login credentials.
- A Python IDE with Python 3.10 environment (*Visual Studio Code* or *PyCharm*)
- You will also need to download and unzip this GitHub repository: *https://github.com/elenalowery/generative-ai*
- Click the **Code** button and select **Download ZIP**





- Unzip the downloaded zip file. In the lab we will refer to this folder as the *git repo* folder.

Generative AI and Large Language Models

Generative AI is a new domain in AI which allows users to interact with models using natural language. A user sends requests (*prompts*) to a model, and the model generates the response. To an end user Generative AI may look like a chatbot or a search engine, but implementation of Generative AI is different from legacy chatbots that rely on hardcoded business rules and search engines that use indexing.

Generative AI uses Large Language Models (LLMs) to generate response to a prompt.

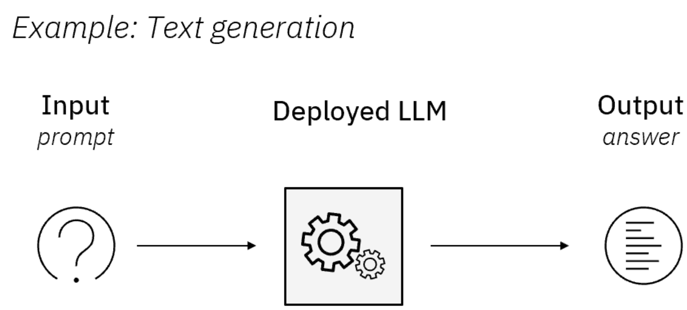
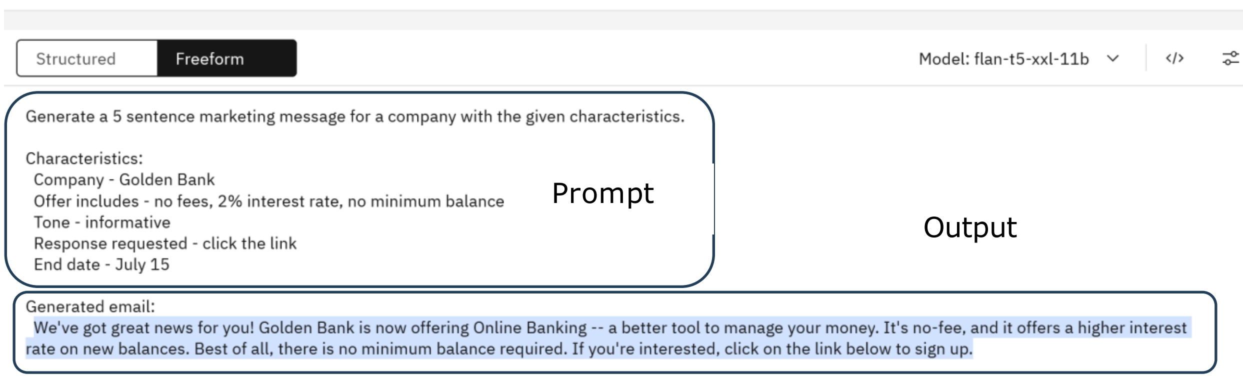


Figure 1: Prompt Lab in watsonx.ai



Unlike traditional machine learning models, which always require training, LLMs are pretrained on a dataset. There are dozens of LLMs, which are developed by different companies. Some companies contribute their models to open source, and many of them are available on the LLM community site *Hugging Face*. It's up to the LLM developer to publish information about the model and the dataset that the model has been trained on. For example, see the model card for one of the popular open source models, *Flan-T5-xxl*. In general, all LLMs are trained on publicly available data. Some companies, including IBM, curate the data used for model training to remove hateful, abusing, and profane (HAP) information.

You can think of the data that the model has been trained on as its "knowledge". For example, if the model was not trained on a dataset that contained 2022 Soccer World Cup results, it will not be able to generate valid/correct answers related to this event. Of course, this applies to all business use cases in which we need models to interact with proprietary enterprise data. We will explain how to solve this problem later in the lab.

Two more factors influence LLM capabilities: size and instruction-tuning. Larger models have been trained on more data and have more parameters. In the context of LLMs, the number of parameters refers to the number of adjustable weights in the neural network that the model uses to generate responses. Parameters are the internal variables of the model that are learned during the training process and represent the knowledge the model has acquired.

While it may seem obvious that a larger model will produce better results, in a production implementation we may need to consider smaller models that meet our use case requirements because of the hosting and inferencing cost.

Instruction-tuned models are models that have been specifically trained for tasks such as classification or summarization. See [IBM documentation] for other considerations when choosing a model.

As we work through the lab, we will introduce a few more important LLM concepts.

Part 1: Understand LLM capabilities

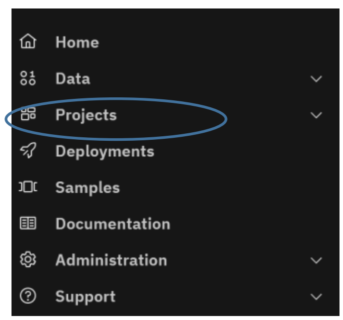
In this lab we will use the **Prompt Lab** in *watsonx.ai* to interact with LLMs included in the platform.

Typically users (prompt engineers or data scientists) have three goals in this phase of the LLM lifecycle:

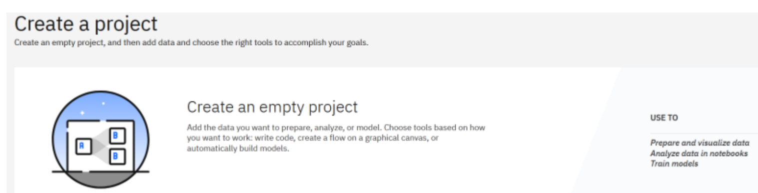
- Find if LLMs can be used for the proposed use case
- Identify the best model and parameters
- Create prompts for the use case.

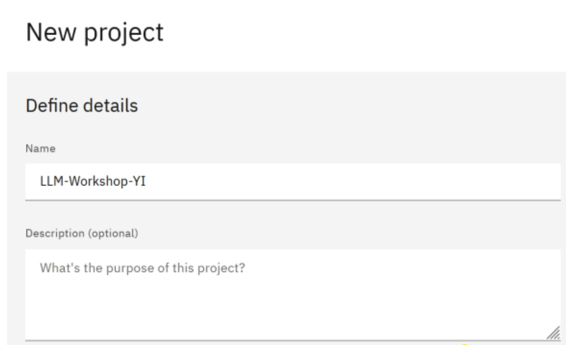
1. Log in to your **IBM Cloud account**. Your lab instructor will provide the URL and userid/password.

2. From the main menu in the top left corner select **Projects -> View All Projects**.

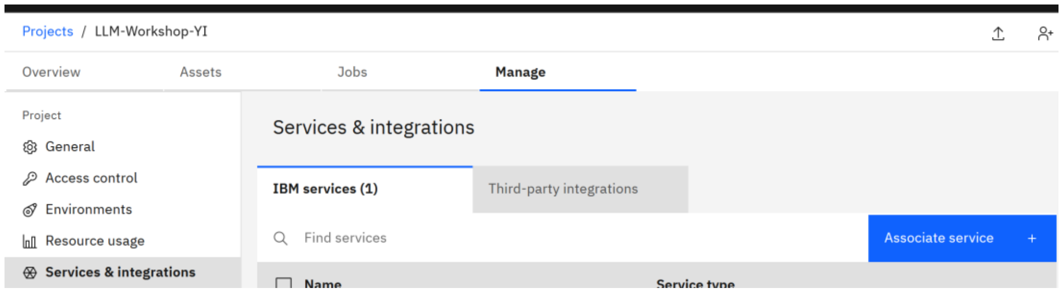


3. Click on the **New Project** button. Select **Empty Project** and add your initials to the project name. For example, *LLM-workshop-YI*.

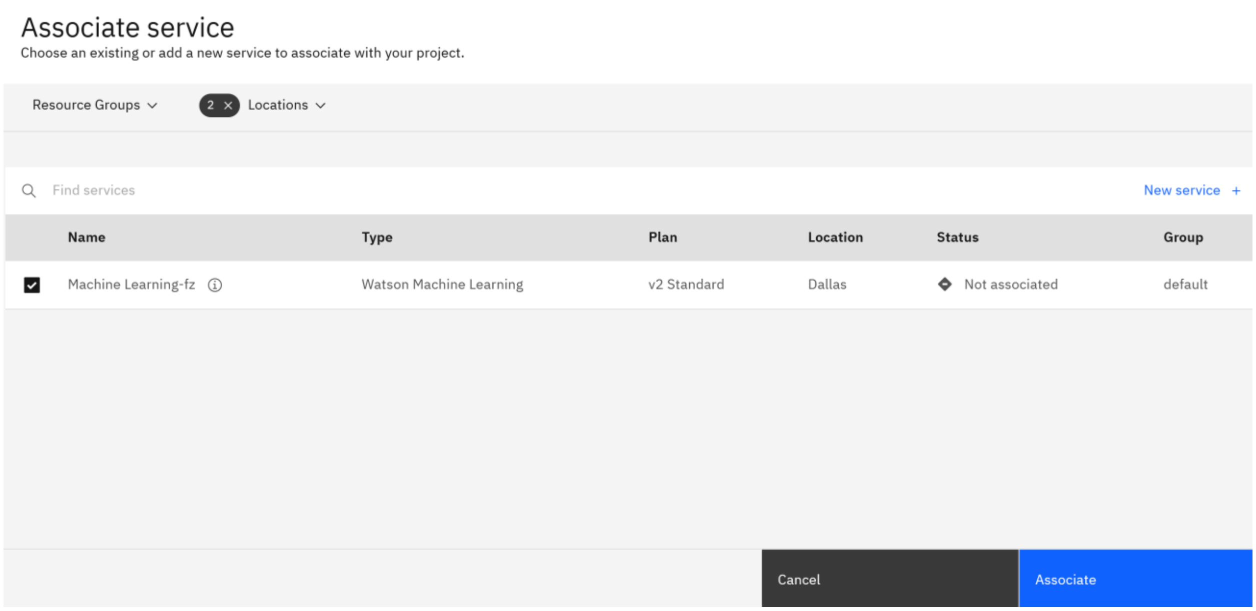




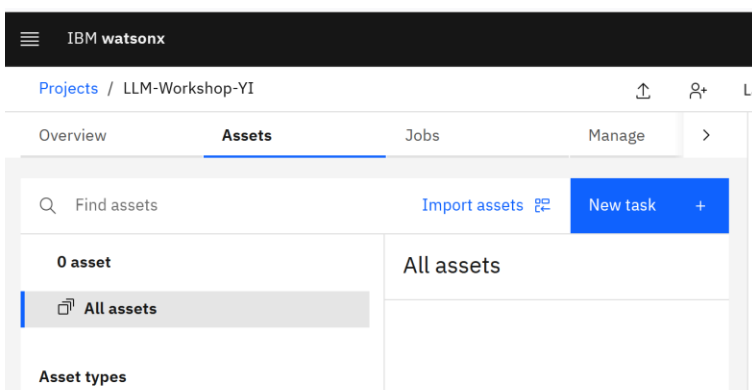
4. Switch to the **Services and Integrations** tab, then click **Associate Service**.



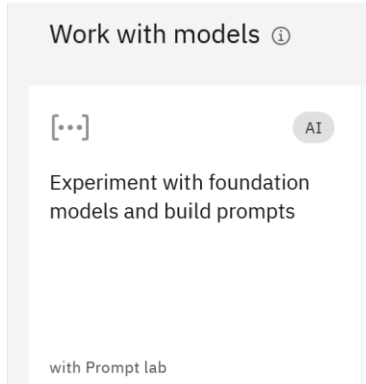
5. Select the displayed **Machine Learning** service and click **Associate**.



6. Switch to the **Assets** tab, then lick the **New task** button.



7. Click on the **Work with models -> Experiment with foundation models…** tile.



Before we start experimenting with prompting, let's review some key concepts about **model selection**.

As you try different models, you will notice that some models return better results with zero-shot prompting. Typically models that have gone through fine-tuning, instruction-tuning, and RLHF provide significantly better output.

- *Fine-tuning* means that the original LLM was trained with high quality labeled data for specific use cases. For example, if we want the model to "Act as an IT architect" and generate output, during the fine-tuning process we need to provide labeled data examples of a writing style for IT architecture.
- If the model goes through the instruction-tuning process, then it will be able to generate output without explicit instructions, such as *"Can you answer this question?"* The model will understand that you're asking a question from the context and sentence structure.
- *RLHF* (*Reinforcement Learning from Human Feedback*) is a technique that's used to improve model output based on feedback provided by testers, usually domain experts (for example, lawyers for generation of legal documents). Before the model is released, it's updated based on testing results.

While all vendors can say that their model has been fine-tuned, instruction-tuned, and has gone through RLHF, since there's no standard process, measurement, or benchmark for these tasks, the quality of the output for these models will vary. The solution to this issue is the broader LLM community. The LLM community is very active and information about the quality of models usually becomes widely known through various community resources such as articles, blogs, and YouTube videos. This is especially true for open source vs. proprietary models. For example, search for "llama vs. ChatGPT" and review results. You can also check documentation provided by the model vendor.

At this time, *llama-70b-2-chat* model, is one of the best models for zero-shot prompting. While it may seem like an obvious choice to always use *llama-70b-2-chat* in watsonx.ai, it may not be possible for several reasons:

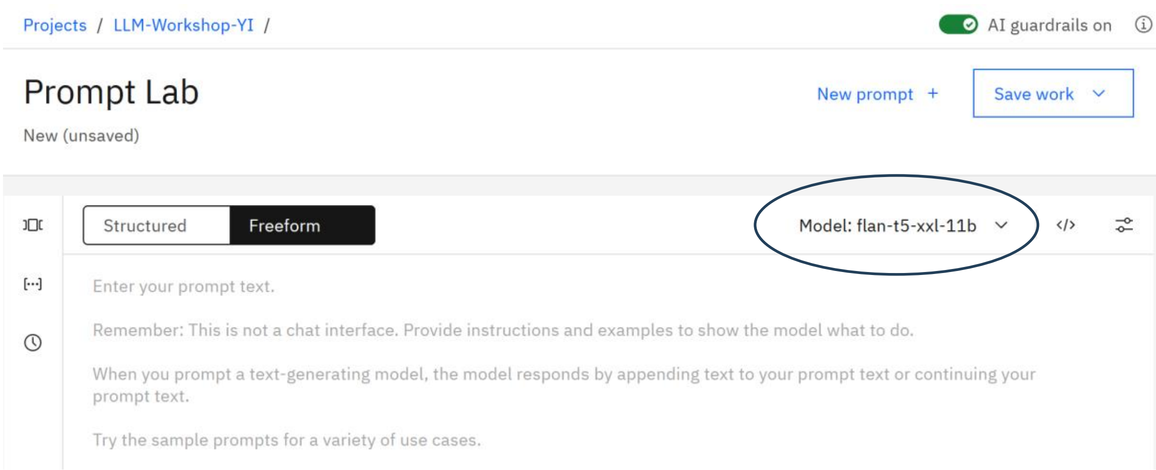
- Model availability in the data center (due to resources or licensing)
- Inference cost
- Hosting cost (for on-premise or hybrid cloud deployments).

It may be possible to achieve similar results with other models or with smaller versions of *llama* by using few-shot prompting or fine-tuning, that's why it's important to experiment with multiple models and understand prompt/turning techniques.

Note: Instructions in this lab are written for the flan model, which is available in all IBM Cloud data centers where watsonx.ai is hosted and in the on-premise version of watsonx.ai. We encourage your to try other models (llama, granite), if they're available in your workshop environment.

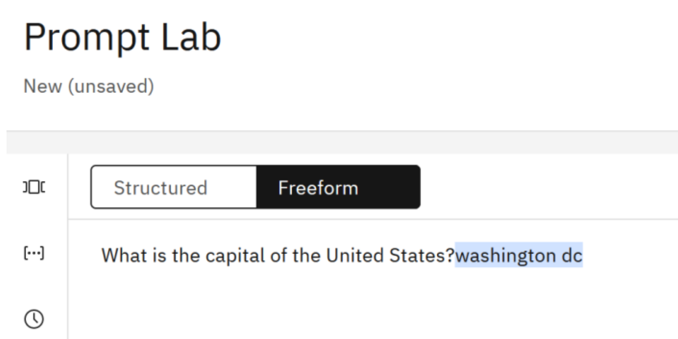
8. After the **Prompt Lab** UI opens, switch to the **Freeform** tab.

   Notice that by default the *flan-t5-xl-11b* model is selected. We will review model settings later in the lab.



Since most LLMs, including the selected *flan* model were trained on publicly available data, we can ask it some questions.

9. Type in the question: *"What is the capital of the United States?"* and click **Generate**. The generated answer is highlighted in blue.



We got the answer to our question without instructing the model to do it because the *flan* model was *instruction-tuned* to answer questions. *Google*, the creator of this model, published the training instructions that were used for the model in this git repository. As you can see in documentation, the instructions are often shown in a "technical format", but they are still helpful for understanding the best prompting options for this model.

Scroll down to the *natural questions* section of the git page. Here we can see the various phrases we can use with the model when asking questions.

```
"natural_questions": [
    ("Question: {question}?\nAnswer:", "{answer}"),
    ("{question}?", "{answer}"),
    ("Answer the following question:\n\n{question}", "{answer}"),
    ("Answer this question:\n\n{question}?", "{answer}"),
    ("Please answer this question: {question}", "{answer}"),
    ("Answer the question...{question}?", "{answer}"),
    ("What is the answer to this question? {question}", "{answer}"),
    ("Can you tell me the answer to {question}?", "{answer}"),
    ("Next question: {question}", "{answer}"),
    ("Q: {question} A:", "{answer}"),
],
```

Next, we will use more detailed instructions. Enter this prompt in the **Prompt Lab**:

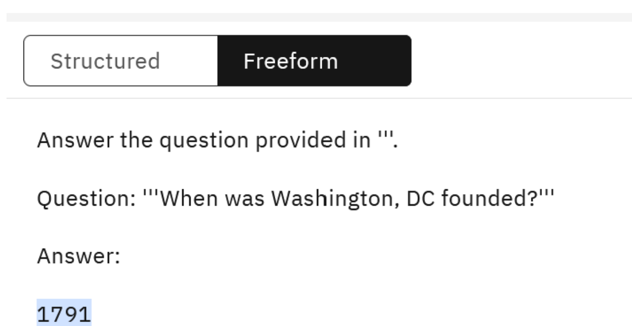
Answer the question provided in '''.

Question: '''When was Washington, DC founded?'''

Answer:

Let's review why we constructed the prompt in this format:

- Triple single quotes (''') are often used to identify a question or text that we want the LLM to use. You can choose other characters, but avoid " (double quotes) because they may already be in the provided text
- Notice that we provided the word "*Answer*:*"* at the end. Remember that LLMs *generate the next probable word"*, and providing the word "Answer" is close to the outcome that we need.



Double check if this is a correct answer by doing a traditional Internet search. You will find out that the correct answer is *July 16th, 1790*. We provided this example to highlight the fact that most LLMs will need to be thoroughly tested before they're deployed into production applications because they do not guarantee 100% correct responses.

One of the ways to improve accuracy of responses is to use *Retrieval Augmented Generation (RAG)* with LLMs. With RAG, we ask the LLM to answer questions or generate content based on the document that we provide. In case of our example, we could give it a *Library of Congress* web page about Washington, DC, and get the correct answer.

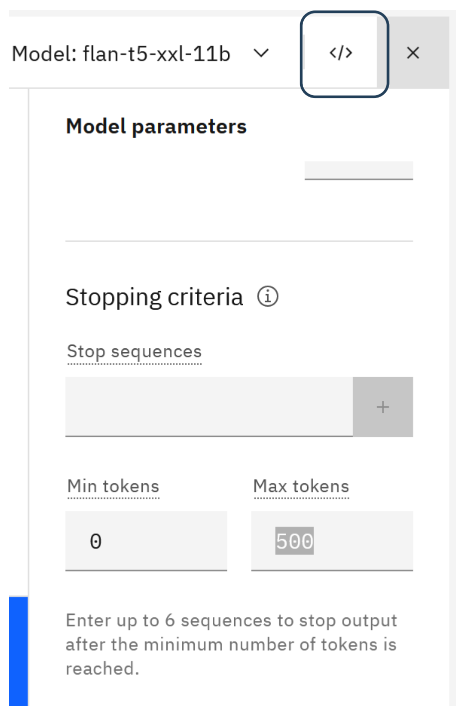
RAG is supported by *watsonx.ai*, and it will be explained in one of the other labs.

Next, we'll test prompts that generate output.

10. Click on model parameters icon in the top right corner.

    If you would like to learn more about each input in the Model parameters panel, you can review documentation.

    Change the **Max tokens** to *500*. When LLMs process instructions and generate output, they convert words to tokens (a sequence of characters). While there isn't a static ratio for letter to token conversion, we can use *10 words = 15 to 20* tokens as a rule of thumb for conversion.

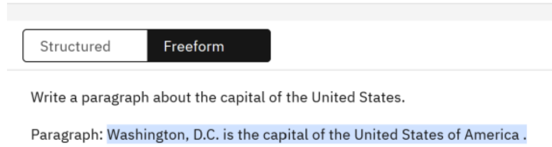


11. Change the prompt to write a paragraph:

    Write a paragraph about the capital of the United States.

    Paragraph:

Notice that our output is rather brief.

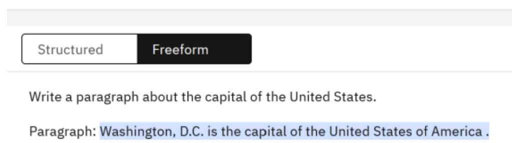


Next, we will try different model parameters and models to see if we can get better results.

12. In the model settings switch **Decoding** from *Greedy* to *Sampling*. Sampling will produce greater variability/creativity in generated content (see [documentation]() for more information).

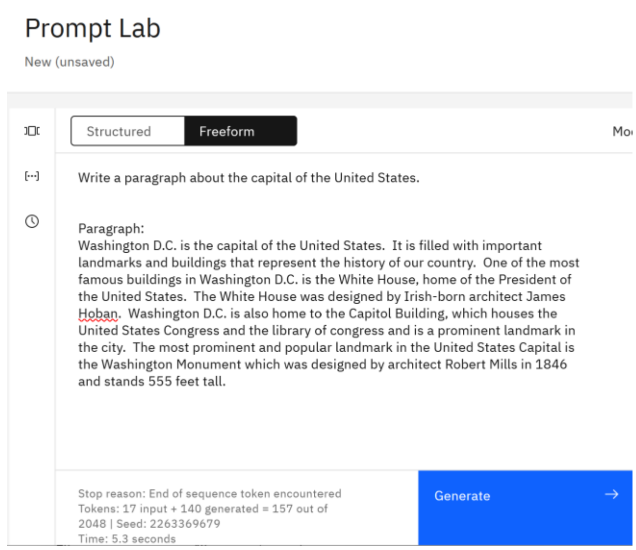
    *Important: Make sure to delete the generated text after the word "Paragraph"**:** before clicking **Generate** again because the model will continue generating after any given text in a prompt, which may result in repetition.*

    Click the **Generate** button. It looks like we're not getting better results with this model, so let's try another one.



13. In the models dropdown select the *mpt-7b-instruct-2* model. Delete the generated text and test again.

    In our testing, we get better results with this model. Notice that every time you click **Generate** (after deleting the generated text), you get different results. We're seeing this because we set the *Decoding* option to *Sampling*. You can also try *Sampling* with different temperature (higher value will result in more variability).



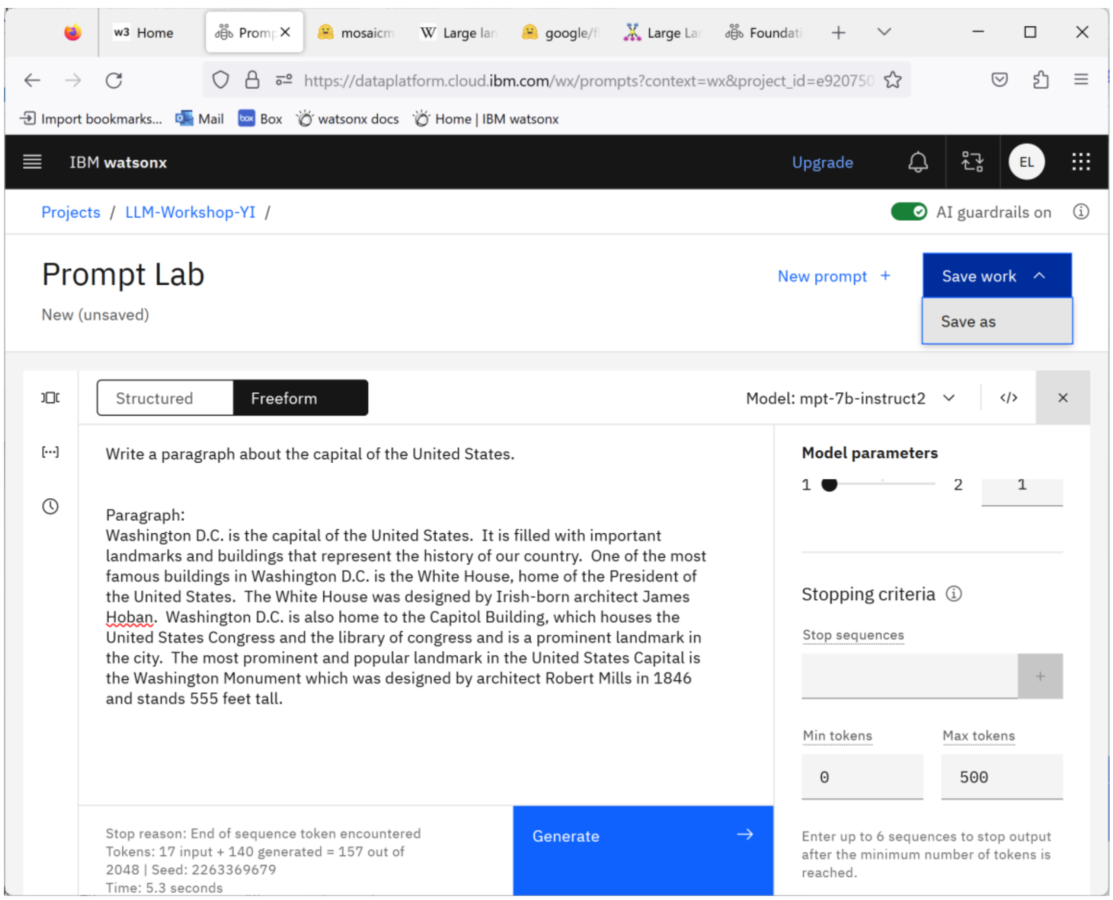
While it may seem unusual that the model generates a different output each time, it's what we're instructing the model to do by both giving it instructions ("*write*") and setting model parameters ("*sampling*"). We would not use the same instructions/parameters for a classification use case which needs to provide consistent output (for example, *positive*, *negative*, or *neutral* sentiment).

You may have already noticed that working with LLMs requires experimentation. In the **Prompt Lab** we can save the results of our experimentation with prompts

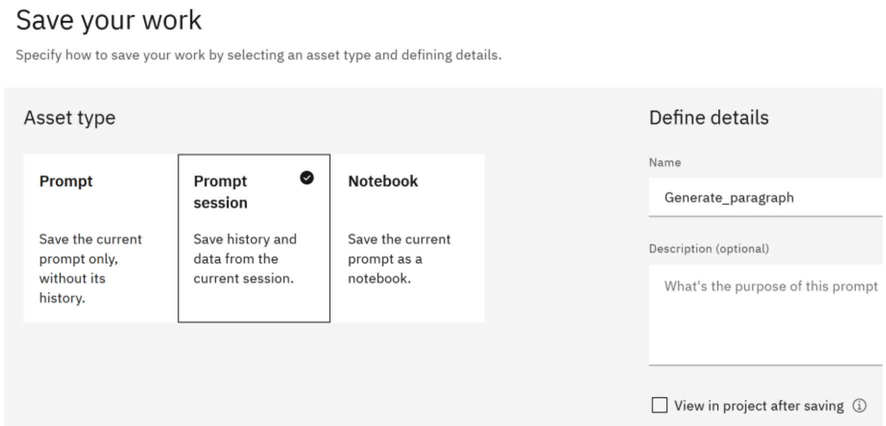
- As a *notebook*
- As a *prompt*
- As a *prompt session*.

If we save our experimentation as a prompt session, we will be able to access various prompts and the output that was generated.

In the **Prompt Lab**, select **Save work -> Save as -> Prompt session**.

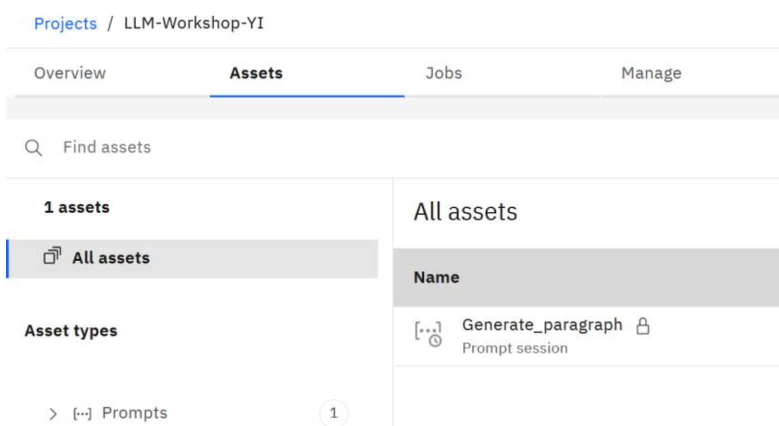


Name the prompt session *Generate_paragraph*.

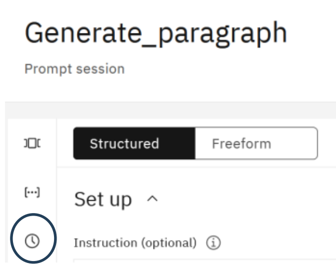


14. Open **watsonx.ai** in another browser window and navigate to your project.

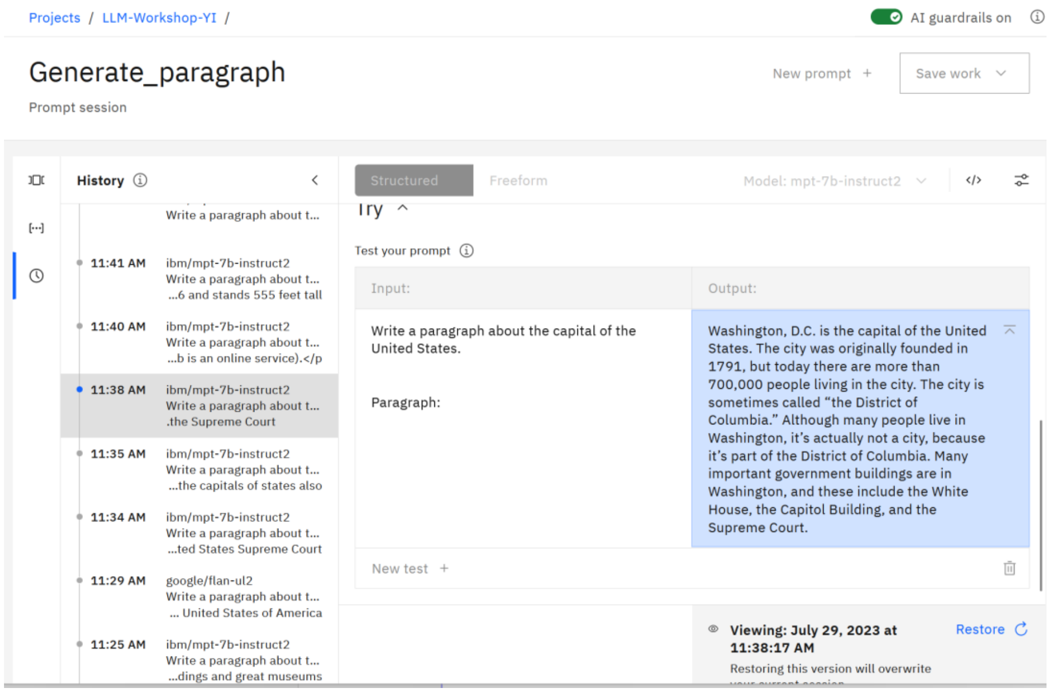
15. Click on the **Assets** tab and open the prompt session asset that you created.



16. In the **Prompt Lab**, click on the *History* icon.



Notice that you can click on various prompts that you tested and view the output in the **Test your promp**t section of the **Prompt Lab**.



Close this browser tab and return to the **Prompt Lab**.

What you have tried so far is a "*question and answer and generation use case with zero-shot prompting*" – you've asked the LLM to generate output without providing any examples.

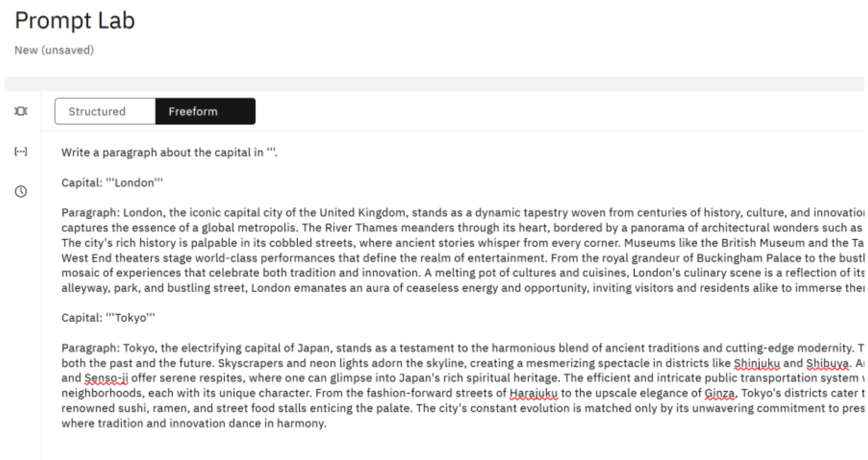
LLMs can be used for several use cases. The most common use cases are:

- Content *summarization*
- Content *classification*
- Content *generation*
- Content *extraction*
- Question *answering.*

The majority of LLMs produce better output when they're given a few examples. This technique is called "few shot learning". The examples are provided in the prompt after the instruction.

Let's test *few shot prompting*.

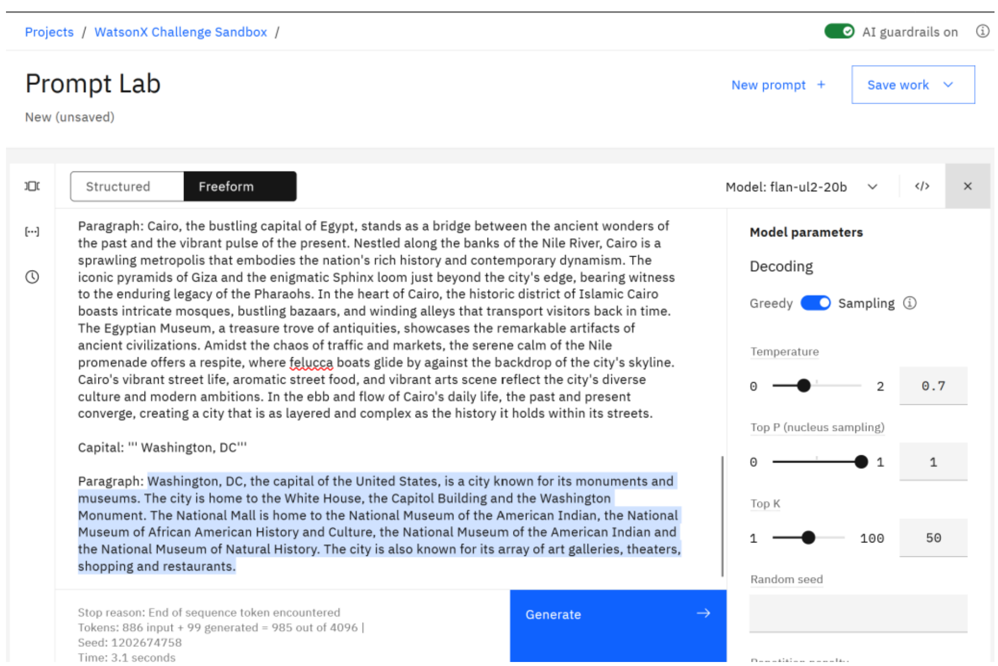
17. In the **Prompt Lab**, create a new prompt. Switch to **Freeform**, and paste the following prompt (this prompt is available in the *Paragraph_few_shot.txt* file in the */Prompts* folder).



18. Modify model parameters:

- Change *decoding* to *sampling* (for more creative output)
- Change the min and max number of tokens to the output you would like to see (for example, *50 min* and *500 max*)
- If you wish, you can test different models. In our experience the *flan* and the *mpt* models that we used earlier in the lab product the best results.

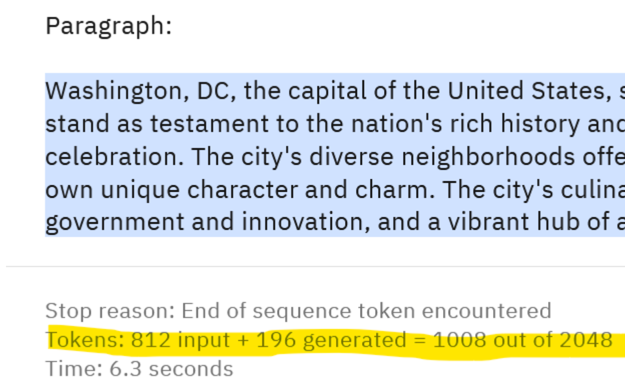
Test the model and review the output.



Next, we will review the concept of *tokens*.

19. Notice the token count that's shown on the bottom of the model output.

    In this screenshot of the *mpt* model output, the "*out of*" number (*2048*) shows the maximum number of tokens that can be processed by a model. If you test with a different model, the maximum number of tokens will be different.



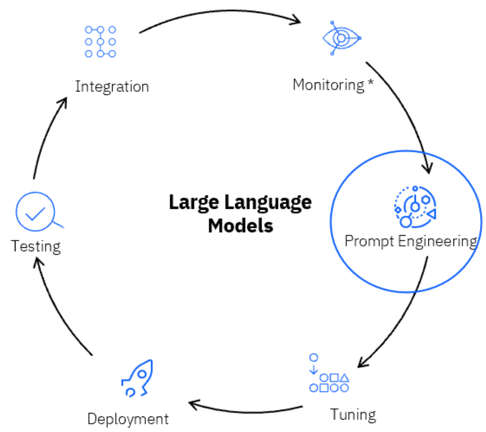
It's important to understand the following facts about tokens:

- All LLMs have a limit for the number of supported tokens. The maximum number of tokens is usually captured in documentation or in the UI, as you've seen in the **Prompt Lab**
- The maximum number of tokens includes both input and output tokens. This means that you can't provide an unlimited number of examples in the prompt
- Some vendors have daily/monthly token limits for different plans, which should be considered when selecting an LLM platform.

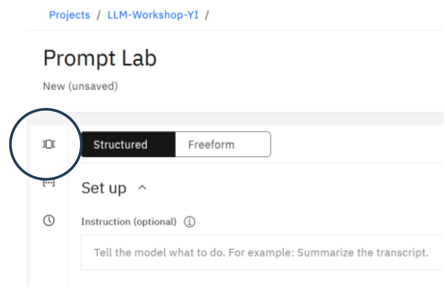
Understanding token constraints is especially important for summarization, generation, and Q&A use cases because they may require more tokens than classification, extraction, or sentiment analysis use cases.

The token constraint limitation can be solved with several approaches. If we need to provide more examples to the model, we can use an approach called *Multitask Prompt Tuning (MPT)* or *fine tuning*. We are not covering these advanced approaches in this introductory lab.

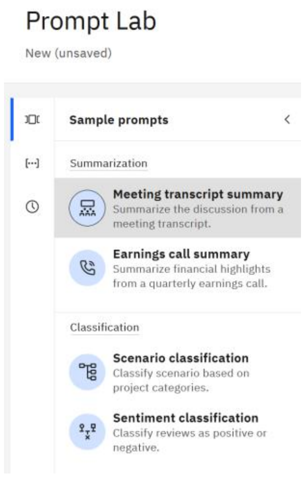
Up to this point you have completed a simple prompt engineering exercise. Prompt engineering is just one of the steps in the process of integrating LLMs into business applications. Before we explore deployment and integration steps, let's review a few sample use cases that can be implemented with LLMs.



20. In the **Prompt Lab**, create a new prompt and click on the **Sample prompts** icon.



21. Explore/test the different prompts, which are organized by use cases.



Notice that some examples use few-shot prompting (*Meeting transcript summar*y), while others (*Sentiment classification*) use zero-shot prompting.

If you wish, try a few sample prompts that we provided in the downloaded git repo */Prompts* folder.

Now that you have reviewed and created prompts, we will test the integration of LLMs with client applications.

Part 2: Integrate LLMs with applications

Many people are familiar with *ChatGPT*, a popular personal assistant application developed by *OpenAI*. Sometimes the terms *generative AI* and even *LLM* are used interchangeably with *ChatGPT*, but *ChatGPT* is more than an LLM, it's a complex application that uses LLMs.

LLMs are building blocks or components of an application, and by themselves they can rarely be used by a business user.

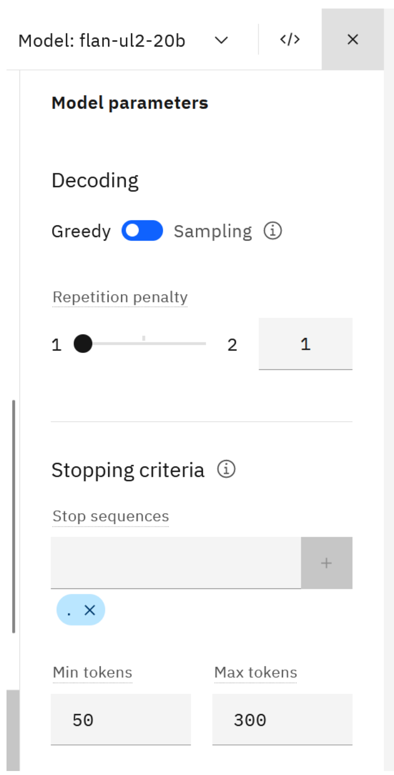
In this section we will explore a simple client application that uses an LLM, then we will provide examples of business applications that can be integrated with LLMs.

1. Navigate to the **Prompt lab** and open one of the prompts you previously created or one of the sample prompts.

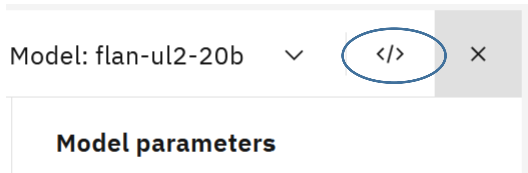
   In this lab we will be using the loan summary prompt which you can find in */Prompts/Loan_few_shot_summary.txt.*

2. Generate a response using this prompt.

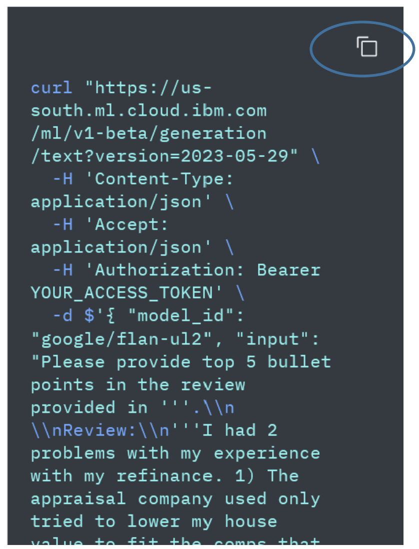
   - You can use either the *flan* of the *mpt* model
   - Keep the decoding method as *Greedy*
   - Add a stop sequence of "." to prevent output that ends with mid-sentence.
   - Make sure to set the min and max tokens to 50 and 300.



After testing the prompt (click **Generate**), click on the **View code** icon.



Copy the code to a notepad.



Let's review the code.

This code is an example of a REST call to invoke the model. **Watsonx.ai** also provides a Python API for model invocation, which we will review later in this lab.

The header of the REST request includes the URL where the model is hosted and a placeholder for the authentication token. At this time all users share a single model inference endpoint. In the future, IBM plans to provide dedicated model endpoints.

Note: IBM does not store model inference input/output data. In the future, users will be able to opt in to storing data.

Security is managed by the *IBM Cloud authentication token*. We will get this token shortly.

The body of the request contains the entire prompt.

File Edit View

Header

```
curl "https://us-south.ml.cloud.ibm.com/ml/v1-beta/generation/text?version=2023-05-29" \
  -H 'Content-Type: application/json' \
  -H 'Accept: application/json' \
  -H 'Authorization: Bearer YOUR_ACCESS_TOKEN' \
  -d $'{
```

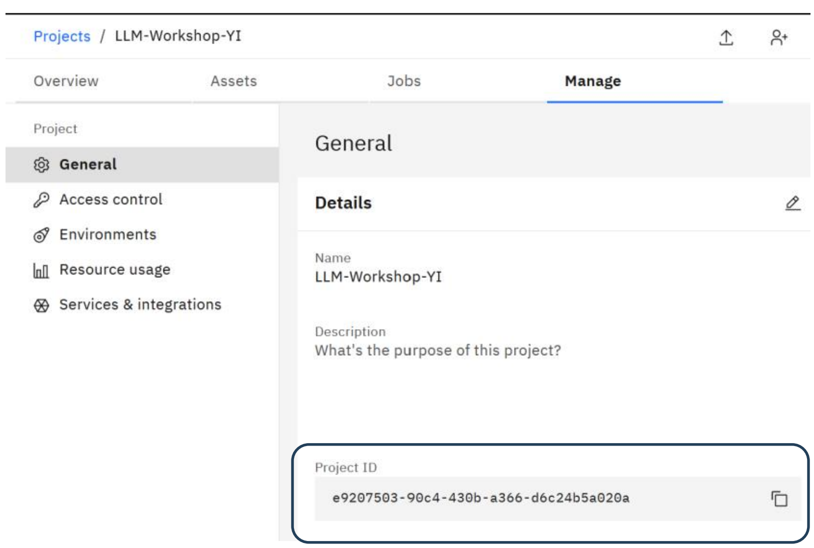
Body

```
  "model_id": "google/flan-ul2",
  "input": "Please provide top 5 bullet points in the review provided in '''.\\n\\nReview:\\n'''I h
experience with my refinance. 1) The appraisal company used only tried to lower my house value to f
to find in the area. My house is unique and he did not use the unique pictures to compare value. He
the appraisal. 2) I started my loan process on a Thursday. On Saturday I tried to contact my loan o
American Express offer that I wanted to apply for. I was informed that it was too late and I could
delay the process. I had just received the email about the offer and I had ju      ed the process
get in on the $2,000 credit on my current bill. I let it go but I should have      the process a
would have helped me out with my bill.'''\\n\\nTop bullet points:\\n1. The appraisal company underv
purposely excluding unique pictures that would have accurately assessed its value.\\n2. The uniquen
taken into consideration, and the appraiser relied solely on comps that did not reflect its true wo
attempted to inform their loan officer about an American Express offer they wanted to apply for, wh
$2,000 credit on their current bill.\\n4. The loan officer stated it was too late to take advantage
delay the process, despite the reviewer having just received the email and recently started the loa
regrets not dropping the process and restarting it to benefit from the offer, as it would have help
\\nReview:'''\\nFor the most part my experience was very quick and very easy. I did however, have 3
```

Finally, at the end of the request we specify model parameters and the *project id*.

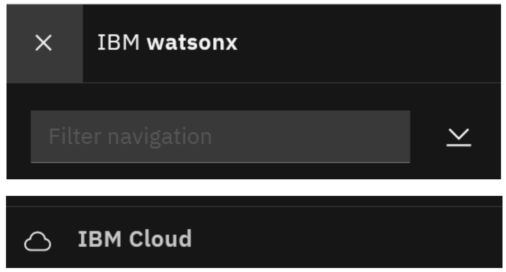
```
  "parameters": {
    "decoding_method": "greedy",
    "max_new_tokens": 300,
    "min_new_tokens": 50,
    "stop_sequences": [
      "."
    ],
    "repetition_penalty": 1
  },
  "project_id": "e9207503-90c4-430b-a366-d6c24b5a020a"
}'
```

The *project id* can be looked up in the **Project -> Manage view** tab of the watxonx.ai project

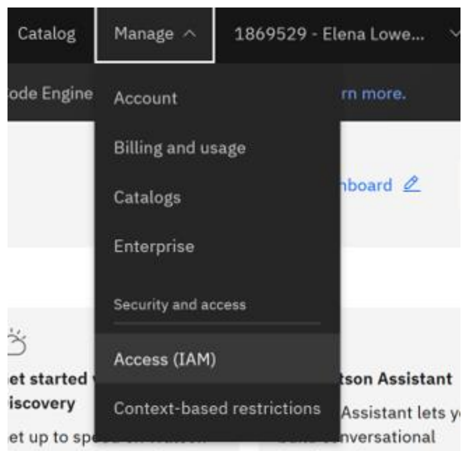


Now we will get the authentication token.

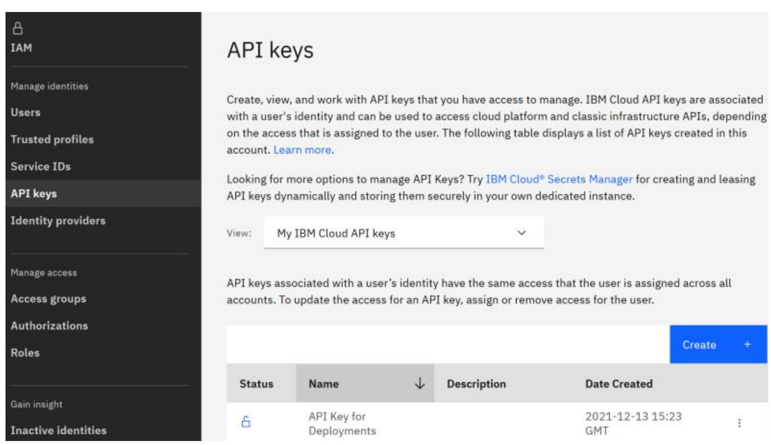
3. Open a new browser window and from the main **watsonx menu** (top right corner), select **IBM Cloud**.



4. Select **Manage -> Access (IAM).**



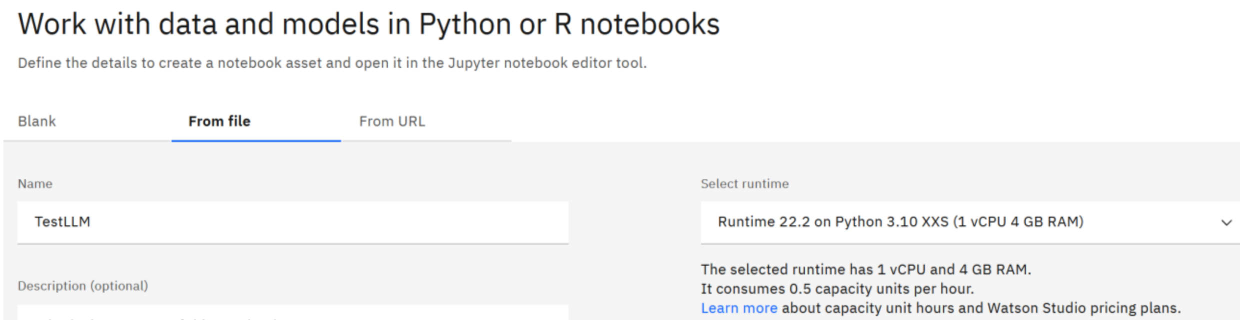
5. Click **API Keys -> Create**. Give the token a name and save it in a notepad. You will use it in the sample notebook.



6. In **watsonx.ai** click **New Task -> Work with data and models in Python or R notebooks**.

Click the **From file** tab and navigate to the downloaded git repo */Notebooks* folder to select the *TestLLM* notebook. Make sure that the *Python 3.10 environment* is selected.

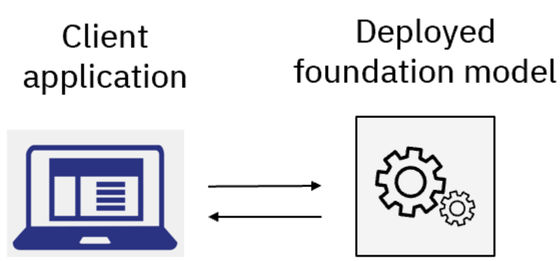
Click **Create** to import the notebook.



Let's review the sample notebook.

This notebook acts as a *client application* that invokes the deployed LLM with a Python SDK. We are using the notebook as a client for simplicity of testing during this lab.

Enterprise client applications can be implemented in Python, Java, .Net and many other programming languages. As discussed earlier, LLMs deployed in **watsonx.**ai can be invoked either with REST calls or with the Python SDK.



Run the notebook to test the LLM with your prompts. See specific instructions in the notebook.

Next, we will use a Python IDE, such as **Visual Studio** or **PyCharm** to run the client application.

22. Find the following Python scripts in the downloaded git repo */Applications* folder:

- *demo_wml_api.py*
- *demo_wml_api_with_streamlit.py*

Load these scripts into your Python IDE.

23. In your Python IDE install the *ibm-watson-machine-learning* library. We recommend that you use Python 3.10 environment.

```
pip install ibm-watson-machine-learning
pip install ibm-cloud-sdk-core
```

Let's review the scripts.

demo_wml_api.py is a simple Python script that shows to how invoke an LLM that's deployed in watsonx.ai. Code in this script can be converted to a module and used in applications that interact with LLMs.

The script has the following functions:

- *get_model()*: creates an LLM model object with the specified parameters
- *answer_questions()*: invokes a model that answers simple questions
- *get_list of_complaints():* generates a list of complaints from a hardcoded customer review
- *invoke_with_REST():* shows how to invoke the LLM using the REST API (other functions use the SDK)
- *get_auth_token():* generates the token that's required for REST invocation
- *demo_LLM_invocation():* invokes all other functions for testing

Prior to running the script, add your *Cloud API key* and *project id*.

```python
# Important: hardcoding the API key in Python code is not a best practice. We are using
# this approach for the ease of demo setup. In a production application these variables
# can be stored in an .env or a properties file

# URL of the hosted LLMs is hardcoded because at this time all LLMs share the same endpoint
url = "https://us-south.ml.cloud.ibm.com"

# Replace with your watsonx project id (look up in the project Manage tab)
watsonx_project_id = ""
# Replace with your IBM Cloud key
api_key = ""
```

24. Run the script. The output will be shown in Python terminal.

```
C:\ProgramData\anaconda3\envs\Python310\python.exe C:\Users\1A3030897\PycharmProjects\LLM_Workshop\demo_wml_apiEL.py
--------------------------------------------------------------------------
Question/request: Write a paragraph about the capital of France.
Answer: Paris is the capital of France, the most populous city of the country, and the largest city in the European Union.
--------------------------------------------------------------------------
--------------------------------------------------------------------------
Prompt:
    From the following customer complaint, extract 3 factors that caused the customer to be unhappy.
    Put each factor on a new line.

    Customer complaint:
            I just tried to book a flight on your incredibly slow website.  All
            the times and prices were confusing.  I liked being able to compare
            the amenities in economy with business class side by side.  But I
            never got to reserve a seat because I didn't understand the seat map.
            Next time, I'll use a travel agent!


    Numbered list of all the factors that caused the customer to be unhappy:
```

Let your instructor know if you have questions.

Next, we will invoke the LLM from a UI. We will use a popular web application development framework, *Streamlit*, to create a simple UI.

You can find the video of this application in the *git repo/Reference* folder.

25. To run this script, you will need to install the *streamlit* package in your Python environment.

    ```
    pip install streamlit
    ```

    ***Important**: If you're running on Windows, you will need to run this script in an Anaconda Python environment because it's the only supported [Python environment] on Windows. Both VS Code and Pycharm can be configured to use Anaconda. Please check with your instructor if you need help with setup instructions.*

26. Open the *demo_wml_api_with_streamlit.py script.* This application uses similar code to invoke the LLM as a previous example.

    The application has 3 functions:
    - *get_model()*: creates an LLM model object with the specified parameters
    - *get_prompt()*: creates a model prompt
    - *answer_questions ():* sets the parameters and invokes the other two functions.

As you can tell by the name of the last function, this is a simple *Question and Answer UI*. You will notice that the prompt is more complicated than the prompt in the previous example: we provide instructions and a few examples (few-shot prompting).

Notice that we are hardcoding the instruction to answer the question. This is just an example, and you can choose to parameterize all components of the prompt.

27. Prior to running the script, add your *Cloud API key* and *project id* as you've done in the previous example.

28. When you run the script, Python will open the *Streamlit UI* in your browser.

    If you invoke Python application from a terminal, and not an IDE then use the following command: `streamlit run demo_wml_api_with_streamlit.py`

    Note: When testing, ask "general knowledge" questions keeping in mind that our prompt is not sophisticated and that the model was trained on generally available data.

Test watsonx.ai LLM

Ask a question, for example: What is IBM?

Which country has the largest population?

Answer to your question: Which country has the largest population? *China, with 1.4 billion people in 2017 (representing 68.1% of the world's population)*

Test watsonx.ai LLM

Ask a question, for example: What is IBM?

What is the hottest temprature ever recorded on Earth?

Answer to your question: What is the hottest temprature ever recorded on Earth? *134 °F (57 °C) in Death Valley, California, United States, on July 10, 1913*

Test watsonx.ai LLM

Ask a question, for example: What is IBM?

Who invented an electric vehicle?

Answer to your question: Who invented an electric vehicle? *Nikola Tesla.. Tesla is credited with inventing the first electric car.. the Tesla Model X.*

Conclusion

You have finished the **Introduction to Generative AI** lab. In this lab you learned:

- Key features of LLMs
- The basics of prompt engineering, including parameter tuning
- Using the **Prompt Lab** to create and test prompts with models available in watsonx.ai
- Testing LLM model inference
- Creating a simple UI to let users interact with LLMs.

What we did not cover in this lab:

- Using LLMs to answer questions on data specific to the enterprise
- Executing multiple instructions with LLMs
- Implementing session management in LLM applications.

If you have questions about these topics, please contact your instructor.