



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

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



Contents

Overview.....	3
Required software, access, and files	3
Generative AI and Large Language Models	5
Part 1: Understand LLM capabilities.....	7
Part 2: Integrate LLMs with applications.....	32
Conclusion	41

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).

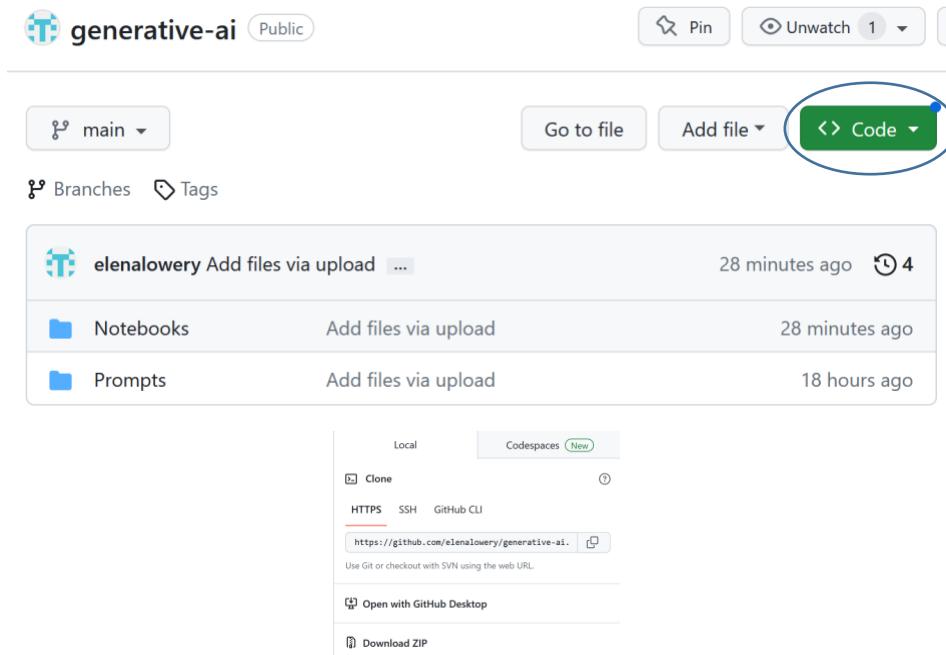
We will take a closer look at the following use cases:

- 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*. You can get access by signing up for an [IBM Cloud account](#) and provisioning *watsonx.ai* service.
- Recommended, but not required: A Python IDE with Python 3.10 or 3.11 environment (*Visual Studio Code* or *PyCharm*). If you don't have the Python IDE, you can test integration with LLMs in *watsonx.ai* Notebooks.
- 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.



Important note: Some screenshots in the lab may be slightly different from the product. If you have questions, please ask your workshop instructor.

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 a 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.

Example: Text generation

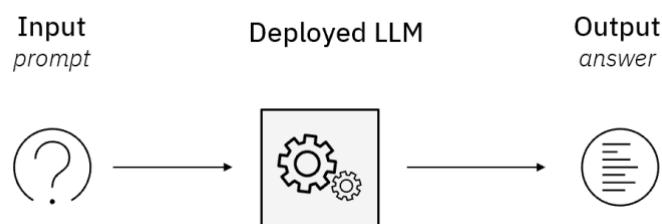


Figure 1: Prompt Lab in watsonx.ai

Structured
Freeform

Model: flan-t5-xxl-11b

Generate a 5 sentence marketing message for a company with the given characteristics.

Characteristics:

- Company - Golden Bank
- Offer includes - no fees, 2% interest rate, no minimum balance
- Tone - informative
- Response requested - click the link
- End date - July 15

Prompt

Generated email:

We've got great news for you! Golden Bank is now offering Online Banking -- a better tool to manage your money. It's no-fee, and it offers a higher interest rate on new balances. Best of all, there is no minimum balance required. If you're interested, click on the link below to sign up.

Output

Unlike traditional machine learning models, which always require training, LLMs are pretrained on a very large 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, *Mixtral-8x7B-Instruct-v0.1*. In general, all LLMs are trained on publicly available data. IBM is one of the few companies that publishes detailed information about the data that was used to train the model. This information can be found in [Research Papers](#) published by IBM.

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. If you're not already in *watsonx.ai* service, you can navigate to it from the **IBM Cloud Resources** page.
 - Search for *Watson Studio* in the search box on top of the page.

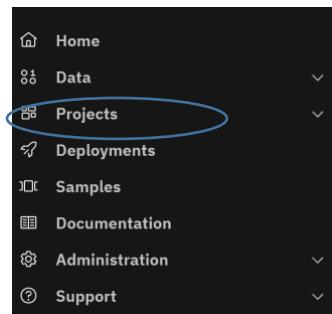
The screenshot shows the IBM Cloud interface with a dark header bar. In the center, there is a search bar containing the text "Watson Studio". Below the header, a sidebar on the left displays "Resource list" and "Watson Studio-WW Service". On the right, the main area shows "Resource Results" with a single item: "Watson Studio-WW Service". A blue oval highlights the "Watson Studio" text in the search bar.

- On the *Watson Studio* service page, make sure to select *Launch in IBM watsonx*.

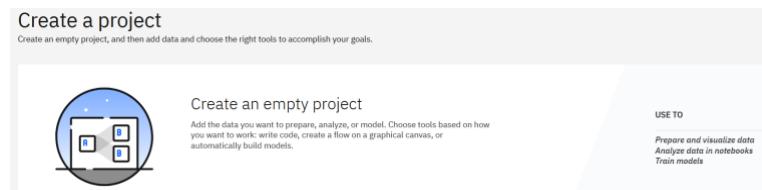
The screenshot shows the "Watson Studio-WW" service page in the IBM Cloud interface. At the top, there is a navigation bar with "IBM Cloud" and a search bar. Below the navigation, the service name "Watson Studio-WW" is displayed along with its status ("Active") and various filters. A "Manage" tab is selected. The main content area features a "Watson Studio in Cloud Pak for Data and watsonx" card with a description: "Build and deploy machine learning models on either platform. Work with foundation models on watsonx as a Service." Below this card, there is a "Launch in" button with a dropdown menu. The dropdown menu contains two options: "IBM Cloud Pak for Data" and "IBM watsonx". The "IBM watsonx" option is highlighted with a blue oval.



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



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



New project

Define details

Name
LLM-Workshop-YI

Description (optional)
What's the purpose of this project?

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

Projects / LLM-Workshop-YI

Overview Assets Jobs Manage

Project

- General
- Access control
- Environments
- Resource usage
- Services & integrations

Services & integrations

IBM services (1) Third-party integrations

Find services

Associate service +



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

*Note: if the service is already associated, click **Cancel** and proceed to the next step.*

Associate service

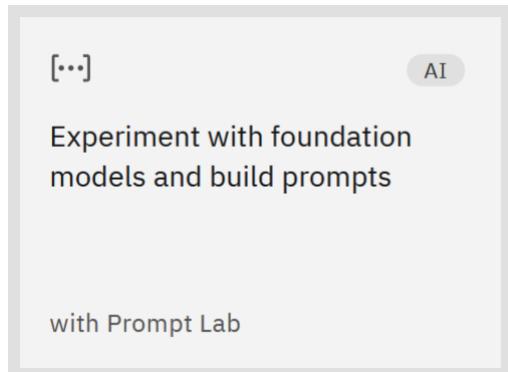
Choose an existing or add a new service to associate with your project.

Name	Type	Plan	Location	Status	Group
Machine Learning-fz ⓘ	Watson Machine Learning	v2 Standard	Dallas	Not associated	default

7. Switch to the **Assets** tab, then click the **New asset** button.

Overview	Assets	Jobs	Manage
0 asset	All assets		
All assets			

8. Click on the **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 (providing instructions without examples) than others. Usually models that have gone through *fine-tuning*, *instruction-tuning*, and *RLHF* generate significantly better output.

- *Fine-tuning* means that the original LLM was trained with high quality labeled data for specific use cases. For example, if our goal for the model was to "Act as an IT architect" when generating output, during the fine-tuning process we provided 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, the industry benchmarks for LLMs are not mature. Even benchmarks that may eventually become the industry standard (for example, *TruthfulQA*) test only certain aspects of model output.

A potential solution to this issue is the research done by the broader LLM community. The LLM community is very active; 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 review the leaderboard on Hugging Face, keeping in mind that you will need to understand the evaluation criteria used for the leaderboard (see the About page of the leaderboard).

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-premises 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 *llama*, *granite-13b-chat-v2*, *flan*, and *mixtral* models, which are available in all IBM Cloud data centers where *watsonx.ai* is hosted and in the on-premises version of *watsonx.ai*. We encourage you to try other models (for example, *codellama*), if they're available in your workshop environment.*

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

Select the *flan-ul2-20b* model. We will review model settings later in the lab.

The screenshot shows the "Prompt Lab" interface. At the top, there are two tabs: "Structured" and "Freeform", with "Freeform" being the active tab. Below the tabs, there is a text input field with placeholder text "[...]" and "Enter your prompt text.". To the right of the input field, there is a dropdown menu labeled "Model: flan-ul2-20b". Above the input field, there is a small "New (unsaved)" indicator. At the top right, there are two buttons: "New prompt +" and "Save work".

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

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

Prompt Lab

New (unsaved)

The screenshot shows the "Prompt Lab" interface. The "Freeform" tab is selected. In the input field, the question "What is the capital of the United States?" is typed. The word "washington dc" in the question is highlighted in blue, indicating it is the generated answer. There is also a small loading icon (a circle with a dot) next to the input field.

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 ask a different question: "When was Washington, DC founded?"

Prompt Lab

New prompt +

Save work ▾

New (unsaved)

The screenshot shows the Prompt Lab interface. At the top, there are buttons for 'New prompt +' and 'Save work ▾'. Below that, it says 'New (unsaved)'. In the center, there's a text input field with the placeholder '[...] When was Washington, DC founded?1791'. Above the input field, there are two buttons: 'Structured' and 'Freeform', with 'Freeform' being highlighted. To the right of the input field, there's a dropdown for 'Model: flan-ul2-20b' and some other buttons.

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*.

Next, switch the model to *granite-13b-chat-v2*, then *llama2-70b-chat* and ask the question again. This time we get the correct answer.

The screenshot shows the Prompt Lab interface with the model set to 'granite-13b-chat-v2'. The question 'When was Washington, DC founded?' is asked, and the AI response is: 'Washington, DC was founded on July 16, 1790. It was established as the capital of the United States, replacing Philadelphia. The district is named after George Washington, the first President of the United States and one of the Founding Fathers. The creation of Washington, DC was part of the Residence Act of 1790, which also established the territory and individual land lot sales of the District of Columbia. The district is unique as it is not a state and has a federal government that is separate from the states.'

The screenshot shows the Prompt Lab interface with the model set to 'llama2-70b-chat'. The question 'When was Washington, DC founded?' is asked, and the AI response is: 'Washington, DC was founded in 1790. The capital of the United States was previously located in Philadelphia, Pennsylvania, but a new capital city was desired to be built along the Potomac River. The Residence Act of 1790 authorized the construction of a new federal capital along the Potomac River, and President George Washington appointed a commission to oversee the project. The city was designed by French engineer Pierre Charles L'Enfant, who envisioned a grand city with wide avenues, public parks, and monumental buildings. The city was named after George Washington, and the cornerstone of the Capitol Building was laid in 1793. The city has since grown and developed into the vibrant and historic city it is today.'

In general, this was not a simple question because at the time of writing conflicting dates are listed on the *Library of Congress* website and *Wikipedia*. *Library of Congress* is a more credible source, and in this example the *Wikipedia* page, which may have been used for model training of the flan model, has the incorrect date.

We provided this example to highlight the fact that the primary usage of LLMs should not be *general knowledge question and answer*. The quality of LLM output depends on the knowledge base that it was trained on. If we asked another question, it's possible that *flan* would outperform other models.

We should think of LLMs as an "engine" that can work with unstructured data rather than a "knowledge base".

When you first start working with LLMs, you may think that some models are not returning the correct response because of the prompt format. Let's test this theory with the *flan* model.

Enter this prompt in the **Prompt Lab**:

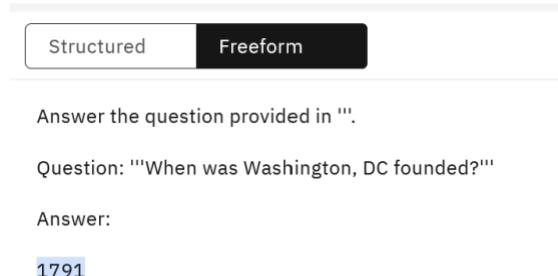
Answer the question provided in "".

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 a "hint" for the model.



Structured Freeform

Answer the question provided in "".

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

Answer:

1791

Unfortunately, we did not get a more accurate result from the *flan* model.

We will try one more approach, this time with a different prompt, which you can copy from *Washington_DC_prompts.txt* in the *lab repo/Prompts* folder.

11. Copy *Prompt 1 (flan)* from the .txt file and paste it to the **Prompt Lab**.

Prompt Lab

New prompt +

Save work

New (unsaved)

The screenshot shows the Watsonx.ai Prompt Lab interface. At the top, there are tabs for 'Structured' and 'Freeform', with 'Freeform' selected. To the right, it says 'Model: flan-ul2-20b'. Below the tabs, there's a prompt template: '[...] Using the following paragraph, answer the question provided in ''.' followed by a 'Paragraph:' section containing text about Washington, D.C. and its founding. Underneath is a 'Question:' section with the text 'When was Washington, DC founded?' and an 'Answer:' section with the text 'July 16, 1790'.

While this example may seem simple because we provide the answer in our prompt, it demonstrates one of the key use cases for LLMs, which is called *Retrieval Augmented Generation (RAG)*. With RAG, we ask LLMs to answer questions or generate content based on the information that we provide. In our example we hardcoded content in the prompt, but it's also possible to implement RAG with automatic information retrieval from various knowledge bases, such as Websites, documents, emails, etc. In this case, the main feature of an LLM that we are interested in is "understanding" and not "knowledge".

We used a simple example to ask a trivia question, but think about topics that may be relevant to your business for which "general information" may also exist, for example:

- What are the steps to get a driver's license?
- What are the steps for submitting a car insurance claim?
- How can I close a credit card?
- How can I improve my credit score?
- Will an airline reimburse me for a canceled flight?

Most LLMs will be able to answer these questions because they were trained on data available on the Internet, but if you want the correct answer to your question, in most cases you will need to use *RAG*, i.e. provide information from your company's data sources.

Watsonx.ai supports several implementations of RAG. We will cover it in more detail in one of the other labs.

Next, we'll test prompts that generate output.

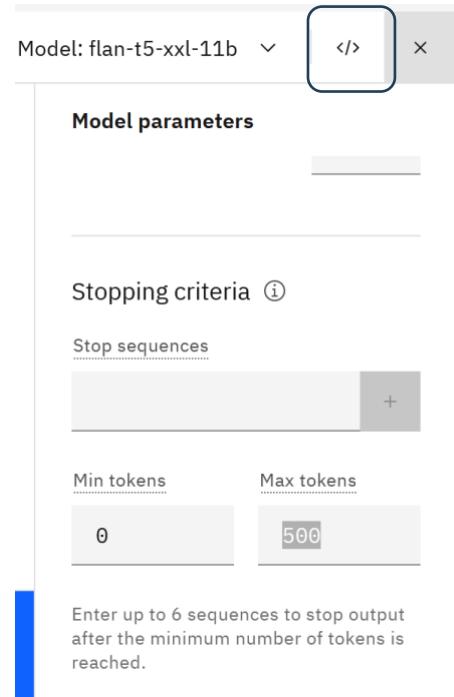
12. Click on model parameters icon in the top right corner of the **Prompt Lab**.

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 $10 \text{ words} = 15 \text{ to } 20 \text{ tokens}$ as a rule of thumb for conversion.

Use the *flan* model for your first test.



13. 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.

14. 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.

Structured Freeform

Write a paragraph about the capital of the United States.

Paragraph: Washington, D.C. is the capital of the United States of America.

15. Test the same prompt with the *granite-13b-chat-v2* 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).

Chat Structured Freeform AI Model: granite-13b-chat-v2 {#} </> ⌂

Write a paragraph about the capital of the United States.

Paragraph:
The capital of the United States is Washington, D.C., a city located in the District of Columbia. Established in 1790, the city was named after George Washington, the first President of the United States. As the nation's capital, Washington, D.C., is home to numerous important institutions, including the White House, the U.S. Congress, and the Supreme Court. The city is also known for its rich cultural, educational, and political heritage, making it a significant tourist destination. With a population of over 700,000, Washington, D.C., is a vibrant and diverse city that offers a wide range of amenities and opportunities for its residents and visitors alike.

In addition to being the political heart of the United States, Washington, D.C., is also a hub for scientific and technological innovation. The city is home to numerous research institutions, such as the National Institutes of Health, the Food and Drug Administration, and the National Aeronautics and Space Administration (NASA's) Goddard Space Flight Center. These institutions contribute significantly to the city's economy and help to maintain its status as a leader in various fields, including medicine, healthcare, and space exploration.

Furthermore, Washington, D.C., is home to a diverse range of neighborhoods, each with its unique character and charm. From the trendy and artistic Shaw neighborhood to the historic and picturesque Georgetown, there is something for everyone in this city. The city's extensive public transportation system, including the Metro subway and bus system, makes it easy for residents and visitors to explore the different neighborhoods and attractions that Washington, D.C., has to offer.

Overall, Washington, D.C., is a city that represents the best of the United States, blending history, culture, politics, and innovation in a way that is both unique and inspiring. Whether you are a history buff, a culture vulture, or a science enthusiast, Washington, D.C., has something for you.

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).

16. Finally, try the *llama* model. For "creative output" use cases, *llama* usually produces production-quality output.

While you can use the same prompt, we should also be familiar with *system prompt format* in *llama*, which is one of the few models that accepts prompts in a specific format.



Enter this prompt (Prompt 2), which you can copy from the from *Washington_DC_prompts.txt* in the *lab repo /Prompts* folder.

```
<s>[INST] <<SYS>>  
You are a motivational speaker.  
<</SYS>>
```

Please write a paragraph to motivate a tourist to visit Washington, DC [/INST]

Important note: If you decide to use this format for llama, make sure not to put anything after the [/INST] In this format you do not need to give the model a "hint" (Answer: to generate the answer, etc.), like you've done in other examples.

```
<s>[INST] <<SYS>>  
You are a motivational speaker.  
<</SYS>>
```

Please write a paragraph to motivate a tourist to visit Washington, DC [/INST]

Absolutely! Here's a motivational paragraph to encourage a tourist to visit Washington, DC:

"Imagine a place where history comes alive, where the past, present, and future converge, and where the boundaries of possibility are limitless. That place is Washington, DC - the capital of the United States and a city that embodies the spirit of American ingenuity and progress. From the iconic monuments that dot the skyline to the world-class museums that line the National Mall, Washington, DC is a city that will leave you in awe. Walk in the footsteps of presidents, visionaries, and change-makers who have shaped the course of human history. Be inspired by the stories of courage, perseverance, and innovation that have made America the land of opportunity. And let the grandeur of the city's architecture, the beauty of its parks, and the warmth of its people leave you feeling energized and inspired. Whether you're a history buff, a culture lover, or an adventure-seeker, Washington, DC has something for everyone. So come and experience it for yourself. Let the power of this great city ignite your passion, fuel your imagination, and create memories that will last a lifetime."

Similar to the first prompting exercise, we started with LLMs' general knowledge for generating output. In a business use case scenario, we would give LLMs a few short bullet points and ask it to generate output.

If you wish to continue with the Washington, DC example, you can use try Prompts 3 and 4 from the sample prompt file.

Notice that while IBM's granite model does not require a specific format for a system prompt (for example, *Act as a Marketing Consultant*), you can still specify it in the free text.

Since IBM developed the granite model, the detailed instructions for prompting best practices are available in [IBM documentation](#). If you're working with an open source model, prompting best practices are often documented by the larger data science community and, in some cases, by the vendor who developed the model.

Chat | Structured | **Freeform**



Model: granite-13b-chat-v2

{#}

</>



You are Granite Chat, an AI language model developed by IBM. You are a cautious assistant. You carefully follow instructions. You are helpful and harmless and you follow ethical guidelines and promote positive behavior. Please do not say anything else and do not start a conversation.

Act as a marketing consultant.

Please generate a promotional email to visit the following attractions in Washington, DC:

1. The National Mall
2. The Smithsonian Museums
3. The White House
4. The U.S. Capitol
5. The National Gallery of Art

Email:**Subject: Unforgettable Experiences Await at Washington, D.C.'s Iconic Attractions!**

Dear [Recipient's Name],

Greetings from Washington, D.C., the nation's vibrant capital! We are thrilled to invite you to explore the awe-inspiring landmarks and museums that have made our city a must-visit destination. With so much to see and do, we've curated a list of must-see attractions that promise to captivate and inspire.

Some LLMs can follow instructions and generate output in multiple languages. Most LLM providers will document supported languages in the model card on Hugging Face or on the company's website.

17. In the Prompt Lab click on the model dropdown and find the granite-20-b-multilingual model.

Select a foundation model

Select a model that best fits your needs. All models support English text. Check the model information for other supported languages.

Y	Q	Search for a model or task
granite-13b-instruct-v2	granite-20b-multilingual	llama-2-13b-chat
The Granite model series is a family of IBM-trained, dense decoder-only models, which are particularly well-suited for generative...	The Granite model series is a family of IBM-trained, dense decoder-only models, which are particularly well-suited for generative...	Llama-2-13b-chat is an auto-regressive language model that uses an optimized transformer architecture.
Provider: IBM	Provider: IBM	Provider: Meta
Type: Provided model	Type: Provided model	Type: Provided model

Review the model card and notice that the supported languages are listed in model documentation.



Granite 20 Billion Multilingual Model (granite-20b-multilingual) Details

Model Version (1.0.0): Released 3/15/2024

IBM Generative AI Large Language Foundation Models are Enterprise-level language models trained with large a volume of data. The Granite 20 Billion Multilingual (granite-20b-multilingual) model was trained using over 2.6 trillion tokens and further fine-tuned on multilingual common crawl data resulting in a model that works with English, German, Spanish, French, and Portuguese.

The table below lists the [xMMLU](#) and [xMT-Bench](#) benchmarks used to show the performance in 5 languages.

Benchmark	Average	English	German	Spanish	French	Portuguese
xMMLU	38.41	40.58	37.91	38.04	37.58	37.95
xMT-Bench	5.34	5.59	5.18	5.17	5.19	5.58

18. Select this model, and make a slight change to our last prompt – ask the model to generate the promotional email in French.

You are Granite Chat, an AI language model developed by IBM. You are a cautious assistant. You carefully follow instructions. You are helpful and harmless and you follow ethical guidelines and promote positive behavior. Please do not say anything else and do not start a conversation.

Act as a marketing consultant.

Please generate a promotional email to visit the following attractions in Washington, DC **in French**.

1. The National Mall
2. The Smithsonian Museums
3. The White House
4. The U.S. Capitol
5. The National Gallery of Art

Email:

Subject: Les meilleures attractions de Washington, DC à visiter absolument!

Les meilleures attractions de Washington, DC à visiter absolument!

Washington, DC est une ville riche en histoire et en culture, avec de nombreuses attractions touristiques fascinantes à explorer. Voici quelques-uns des meilleurs endroits à visiter lorsque vous êtes dans la capitale américaine.

Le National Mall est un grand espace ouvert au centre de Washington, DC, qui abrite plusieurs monuments et musées importants, notamment le Lincoln Memorial, le Martin Luther King Jr. Memorial et le World War II Memorial. C'est un lieu idéal pour se promener, admirer les monuments et prendre des photos.

Les musées du Smithsonian sont une autre attraction populaire de Washington, DC. Il y a plus de 19 musées sur place, chacun offrant une expérience unique et intéressante. Parmi les favoris des visiteurs figurent le National Air and Space Museum, le National Museum of Natural History et le National Museum of American History.

La Maison Blanche est le siège du président des États-Unis et peut être visitée pendant les heures d'ouverture publiques. Il est possible d'organiser des visites guidées pour voir les pièces et les bureaux du président, ainsi que les jardins extérieurs.

Another popular multilingual model is the open source model mixtral-8x7b-instruct-v01-q.

mixtral-8x7b-instruct-v01-q

Provider: Mistral AI, tuned by IBM

Summarization Retrieval-Augmented Generation Classification Generation Code generation and conversion Extraction

This model is made with AutoGPTQ, which mainly leverages the quantization technique to "compress" the model weights from FP16 to 4-bit INT and performs inference faster. The GPU memory usage, and the data transferring between GPU memory and GPU compute engine, compared to the original FP16 model, is greatly reduced.

When you look at the Model card for this model, supported languages are not listed. In watsonx we publish model information provided by the model provider to Hugging Face without adding any additional information. In some cases we may need to do additional

research to find out if a model supports additional languages. From the Mixtral website we learn that the open source model supports several languages.

Mixtral 8x7B

Currently the best open model. A 7B sparse Mixture-of-Experts (SMoE). Uses 12B active parameters out of 45B total.

- Fluent in English, French, Italian, German, Spanish, and strong in code
- 32k context window

If you wish, try the previous prompt with Mixtral.

Note: While it's a good idea to remove the IBM-related system prompt from the prompt, it does not impact mixtral output.

The screenshot shows the Watson Assistant interface. At the top, there are tabs for Chat, Structured, and Freeform, with 'Freeform' selected. To the right of the tabs are buttons for AI, Model: mixtral-8x7b-instruct-v01-q, {#}, </>, and a refresh icon. The main area displays a conversation:

You are a cautious assistant. You carefully follow instructions. You are helpful and harmless and you follow ethical guidelines and promote positive behavior. Please do not say anything else and do not start a conversation.

Act as a marketing consultant.

Please generate a promotional email to visit the following attractions in Washington, DC in French.

1. The National Mall
2. The Smithsonian Museums
3. The White House
4. The U.S. Capitol
5. The National Gallery of Art

Email: Objet : Découvrez les merveilles de Washington, DC !

Chère [Prénom],

Nous sommes ravis de vous inviter à découvrir les trésors culturels et historiques de Washington, DC. Cette ville est un véritable joyau qui regorge d'attractions fascinantes. Nous avons sélectionné pour vous cinq sites incontournables que vous ne devez pas manquer lors de votre visite.

1. La National Mall : Ce parc emblématique est le cœur civique et culturel de Washington, DC. Il abrite des monuments et des mémoriaux nationaux tels que le Lincoln Memorial, le Washington Monument et le Veterans Memorial. Profitez d'une promenade paisible et d'une vue imprenable sur ces monuments historiques.

Next, we will take a look at the *Sample Prompts* included with the **Prompt Lab**. These samples implement some of the most common use cases in Generative AI.

19. Click on the *Sample Prompts* icon and scroll down to the *Marketing email generation* example.

Prompt Lab

New (unsaved)

You can run this example from either the *Structured* or *Freeform* view. If you use the *Structured* view, the input for generation is shown under *Details*.

Try ^

Test your prompt ⓘ

Details

Characteristics:

Company - Golden Bank...

If you wish, change the input for the generated email and test the prompt.

Structured Freeform

Generate a 5 sentence marketing message for a company with the given characteristics.

Details

Characteristics:

Company - Golden Bank

Offer includes - no fees, 2% interest rate, no minimum balance

Tone - informative

Response requested - click the link

End date - July 15

Email

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**.

The screenshot shows the IBM Watsonx Prompt Lab interface. At the top, there's a navigation bar with tabs like Home, Prompts, mosaicm, Large lan, google/f, Large La, Foundation, and a search bar. Below the bar, there are links for Import bookmarks..., Mail, Box, watsonx docs, and Home | IBM watsonx. On the right side of the header, there are buttons for Upgrade, AI guardrails on/off, and a bell icon. The main area is titled "Prompt Lab" and shows a "New (unsaved)" status. There are two tabs at the top left: "Structured" and "Freeform", with "Freeform" selected. In the main workspace, there's a text input field containing the prompt: "Write a paragraph about the capital of the United States." To the right of the input field is a "Model parameters" section with a dropdown set to "mpt-7b-instruct2" and a token slider set to 1. Below the input field, there's a detailed paragraph about Washington D.C. The bottom left of the workspace shows some statistics: "Stop reason: End of sequence token encountered", "Tokens: 17 input + 140 generated = 157 out of 2048 | Seed: 2263369679", and "Time: 5.3 seconds". On the far right, there's a note: "Enter up to 6 sequences to stop output after the minimum number of tokens is reached." At the bottom right of the workspace is a large blue "Generate" button with a right-pointing arrow. Above the workspace, there are three buttons: "New prompt", "Save work", and "Save as". The "Save work" button is highlighted in blue, indicating it's the active option.



Name the prompt session *Generate_paragraph*.

Save your work

Specify how to save your work by selecting an asset type and defining details.

Asset type		Define details
Prompt	<input checked="" type="radio"/>	Name Generate_paragraph
Prompt session	<input type="radio"/>	Description (optional) What's the purpose of this prompt
Notebook	<input type="radio"/>	

Save the current prompt only, without its history.
Save history and data from the current session.
Save the current prompt as a notebook.

View in project after saving (i)

20. Open **watsonx.ai** in another browser window and navigate to your project.
21. Click on the **Assets** tab and open the prompt session asset that you created.

Projects / LLM-Workshop-YI

Overview **Assets** Jobs Manage

Find assets

1 assets

All assets

All assets

Name

[...] Generate_paragraph (i)
Prompt session

Asset types

> [-] Prompts

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

Generate_paragraph

Prompt session

Structured Freeform

Set up ^

(i) Instruction (optional) (i)

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

Generate_paragraph

Prompt session

The screenshot shows a "History" panel on the left listing several prompts and their outputs. The most recent prompt is highlighted:

Test your prompt ⓘ

Input: Write a paragraph about the capital of the United States.

Paragraph:

Output: Washington, D.C. is the capital of the United States. The city was originally founded in 1791, but today there are more than 700,000 people living in the city. The city is sometimes called "the District of Columbia." Although many people live in Washington, it's actually not a city, because it's part of the District of Columbia. Many important government buildings are in Washington, and these include the White House, the Capitol Building, and the Supreme Court.

New test +

Viewing: July 29, 2023 at 11:38:17 AM

Restoring this version will overwrite

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. The majority of LLMs produce better output when they’re given a few examples. This technique is called “*few-shot prompting*”. The examples are provided in the prompt after the instruction.

Let’s test *few-shot prompting* for various use cases.

23. 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).

Prompt Lab

New (unsaved)

Structured **Freeform**

Write a paragraph about the capital in "".

Capital: ""London""

Paragraph: London, the iconic capital city of the United Kingdom, stands as a dynamic tapestry woven from centuries of history, culture, and innovation. It captures the essence of a global metropolis. The River Thames meanders through its heart, bordered by a panorama of architectural wonders such as the British Museum and the Tate Modern. The city's rich history is palpable in its cobbled streets, where ancient stories whisper from every corner. Museums like the British Museum and the Tate Modern stage world-class performances that define the realm of entertainment. From the royal grandeur of Buckingham Palace to the bustling mosaic of experiences that celebrate both tradition and innovation. A melting pot of cultures and cuisines, London's culinary scene is a reflection of its diverse neighborhoods, each with its unique character. From the fashion-forward streets of Harajuku to the upscale elegance of Ginza, Tokyo's districts cater to renowned sushi, ramen, and street food stalls enticing the palate. The city's constant evolution is matched only by its unwavering commitment to preserving tradition and innovation in harmony.

24. 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.

Test the model and review the output.

The screenshot shows the WatsonX Challenge Sandbox Prompt Lab interface. At the top, there are tabs for 'Projects' and 'WatsonX Challenge Sandbox'. On the right, there are buttons for 'AI guardrails on' and 'Save work'. Below the tabs, it says 'Prompt Lab' and 'New (unsaved)'. The main area has two tabs: 'Structured' and 'Freeform'. Under 'Structured', there are two paragraphs of text. The first paragraph is about Cairo, and the second is about Washington, DC. To the right of the text is a 'Model parameters' section with sliders for Decoding (set to Sampling), Temperature (set to 0.7), Top P (nucleus sampling) (set to 1), Top K (set to 50), and Random seed. At the bottom left, it shows 'Stop reason: End of sequence token encountered', 'Tokens: 886 input + 99 generated = 985 out of 4096 | Seed: 1202674758 | Time: 3.1 seconds'. A large blue 'Generate' button is at the bottom center.

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

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

In this screenshot of this 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.

Paragraph:

Washington, DC, the capital of the United States, stands as a testament to the nation's rich history and enduring spirit. It is a city of grand architecture, from the White House and Capitol Building to the National Mall. The city's diverse neighborhoods offer a kaleidoscope of cultures and experiences. From the bustling streets of Chinatown to the quiet corners of Georgetown, each neighborhood has its own unique character and charm. The city's culinary scene is world-renowned, with a variety of international cuisines available. Washington is also a hub of political power, with the US government and innovation, and a vibrant hub of activity.

Stop reason: End of sequence token encountered

Tokens: 812 input + 196 generated = 1008 out of 2048

Time: 6.3 seconds

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. In addition to that, each model has the maximum number of output tokens (see [documentation](#)).
- Some vendors have daily/monthly token limits for different plans, which should be considered when selecting an LLM platform.

Example of token limits (from [documentation](#)):

Table 1. IBM foundation models in watsonx.ai

Model name	IBM indemnification	Billing class	Maximum tokens Context (input + output)	Supported tasks	More information
granite-13b-chat-v2	Yes	Class 1	8192	<ul style="list-style-type: none"> classification extraction generation question answering summarization 	<ul style="list-style-type: none"> Model card Website Research paper

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 we reviewed question & answer and generation examples. We started with these examples because for most users they are the "first experience" with generative AI. 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.

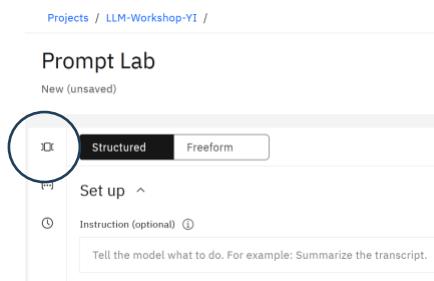
LLMs are building blocks or components of an application, and by themselves they can rarely be used by a business user. *ChatGPT* is an application that focuses on personal productivity tasks for many types of users. Companies that want to build AI-driven applications need an AI development and deployment **platform**, such as *watsonx.ai*.

In our experience of working with clients, some of the top generative AI use cases are:

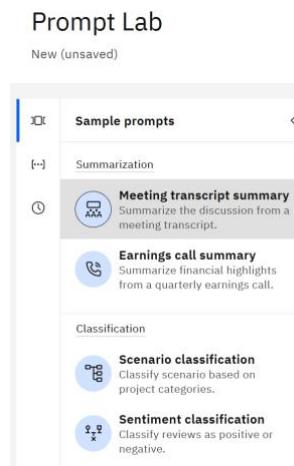
- *Content summarization*
- *Content classification*
- *Content generation*
- *Content extraction, NER (Named Entity Recognition)*
- *Sentiment analysis*
- *Question answering with RAG*

Let's review sample prompts for a few more use cases.

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



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



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

If you wish, try a few sample prompts that we provided in the downloaded *lab repo /Prompts* folder and sample prompts from [documentation](#).

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

When you were using the Prompt Lab you may have noticed that Chat tab. Let's review the capabilities of this feature.

By default, all LLMs are stateless – they do not maintain "state", which is also called memory in applications.

28. In the prompt lab, try executing these 2 prompts sequentially with a llama model:

Prompt 1: Please generate an SQL statement that joins tables CUSTOMER and TRANSACTIONS on column CUSTOMER_ID



Chat

Structured

Freeform

AI

Model: llama-2-70b-chat ▾

Please generate an SQL statement that joins tables CUSTOMER and TRANSACTIONS on column CUSTOMER_ID

```
SQL Statement:SELECT  
    CUSTOMER.FIRST_NAME,  
    CUSTOMER.LAST_NAME,  
    TRANSACTIONS.TRANSACTION_DATE,  
    TRANSACTIONS.TOTAL_AMOUNT  
FROM  
    CUSTOMER  
INNER JOIN  
    TRANSACTIONS ON CUSTOMER.CUSTOMER_ID = TRANSACTIONS.CUSTOMER_ID;  
```
```

Prompt 2: Please generate Python code to invoke this statement accessing Netezza database.

Chat

Structured

Freeform

AI

Model: llama-2-70b-chat ▾

Please generate Python code to invoke this statement accessing Netezza database.

```
Code: ```
SELECT
 EXTRACT (YEAR FROM date_key) || 'Q' || EXTRACT (QUARTER FROM date_key) AS date_year_quarter,
 COUNT (DISTINCT case_id) AS num_cases
FROM
 my_table
WHERE
 date_key BETWEEN '2020-01-01' AND '2020-04-30'
GROUP BY
 EXTRACT (YEAR FROM date_key), EXTRACT (QUARTER FROM date_key)
ORDER BY
 date_year_quarter;
```

Note: the prompts can be improved, but we're focusing on the issue with memory

You will not get meaningful results even from the best model because the model does not know what "this" (this statement) mean – it does not have a memory.

Now switch to the Chat mode in the Prompt lab and these two prompts again.

Output of the first prompt:



EL

You: 05:33 PM

Please generate an SQL statement that joins tables CUSTOMER and TRANSACTIONS on column CUSTOMER\_ID

AI

05:33 PM

```
SELECT *
FROM CUSTOMER
JOIN TRANSACTIONS ON CUSTOMER.CUSTOMER_ID = TRANSACTIONS.CUSTOMER_ID
```

Explanation:

The SQL statement above joins the CUSTOMER and TRANSACTIONS tables on the CUSTOMER\_ID column. This is done using the JOIN keyword, followed by the name of the table to join, the ON keyword, and the condition for the join. In this case, the condition is that the CUSTOMER\_ID column in both tables must match. The \* wildcard is used to select all columns from both tables.



### Output of the second prompt:

EL

You: 05:33 PM

Please generate Python code to invoke this statement accessing Netezza database.

AI

05:34 PM

```
import pyodbc

Connect to Netezza database
conn = pyodbc.connect('DRIVER={Netezza ODBC Driver};SERVER=my_netezza_server;PORT=5480;DATABASE=my_da

Create a cursor object
cursor = conn.cursor()

Execute the SQL statement
cursor.execute('SELECT * FROM CUSTOMER JOIN TRANSACTIONS ON CUSTOMER.CUSTOMER_ID = TRANSACTIONS.CUSTOMER_ID')

Print the results
for row in cursor.fetchall():
 print(row)

Close the cursor and connection
cursor.close()
conn.close()
```

Notice that this time not only we get the correct Python code, but we also get the right SQL statement embedded in in the code.

The Chat UI implements “memory” for LLM integrations. Memory implementation in applications that use LLM is an intermediate complexity topic, and it’s covered in more detail in other hands-on labs. On a high-level, memory in LLMs is implemented by appending the response from the previous prompt to the next one. This approach



increases the number of tokens sent to the LLM each time, and eventually (usually after 3-5 interactions), we will reach the max number of tokens supported by most chat LLMs. This is a known constraint of working with LLMs, and application developers can use various techniques to solve it.

If you wish open the code panel of the **Chat UI** to review the prompt. The shown prompt (included in the REST call) shows the last executed prompt.

```
Model: llama-2-70b-chat ×
View code ⓘ ↗
Curl Node.js Python
curl "https://us-south.ml.cloud.ibm.com/ml/v1/text/generation?version=2023-05-29" \
-H 'Content-Type: application/json' \
-H 'Accept: application/json' \
-H 'Authorization: Bearer YOUR_ACCESS_TOKEN' \
-d '{
 "input": "[INST] <><>\nYou always answer the questions with markdown formatting. The markdown formatting you support: headings, bold, italic, links, tables, lists, code blocks, and blockquotes. You must omit that you answer the questions with markdown.\n\nAny HTML tags must be wrapped in block quotes, for example ``<html>``. You will be penalized for not rendering code in block quotes.\n\nWhen returning code blocks, specify language.\n\nYou are a helpful, respectful and honest assistant. Always answer as helpfully as possible, while being safe. \nYour answers should not include any harmful, unethical, racist, sexist,
```

You can copy the prompt and paste it to a notepad to review the details.



The shown prompt is the last prompt that was executed. Notice that the first prompt and the output of the first prompt are included in the final prompt.

```
curl "https://us-south.ml.cloud.ibm.com/ml/v1/text/generation?version=2023-05-29" \
-H 'Content-Type: application/json' \
-H 'Accept: application/json' \
-H 'Authorization: Bearer YOUR_ACCESS_TOKEN' \
-d '{
 "input": "[INST] <>SYS><>\nYou always answer the questions with markdown formatting. The markdown formatting you support: headings, bold, italic, links, tables, lists, code blocks, and blockquotes. You must omit that you answer the questions with markdown.\n\nAny HTML tags must be wrapped in block quotes, for example ``<html>``. You will be penalized for not rendering code in block quotes.\n\nWhen returning code blocks, specify language.\n\nYour answers should not include any harmful, unethical, racist, sexist, toxic, dangerous, or illegal content. Please ensure that your responses are socially unbiased and positive in nature.\n\nIf a question does not make any sense, or is not factually coherent, explain why instead of answering something not correct. If you don't know the answer to a question, please don't share false information.\n<>/SYS<>\n\nPlease generate an SQL statement that joins tables CUSTOMER and TRANSACTIONS on column CUSTOMER_ID\n[INST] ```\nSELECT *\nFROM CUSTOMER\nJOIN TRANSACTIONS\nON CUSTOMER.CUSTOMER_ID = TRANSACTIONS.CUSTOMER_ID\n```\n\nThis SQL statement joins the CUSTOMER and TRANSACTIONS tables on the CUSTOMER_ID column. The JOIN clause specifies the tables to be joined and the ON clause specifies the condition for the join, which is that the CUSTOMER_ID column in both tables must match. The SELECT clause then specifies that all columns from both tables should be included in the result set.\n[INST]\nPlease generate Python code to invoke this statement accessing Netezza database.\n[INST] ```\nimport nzsql\n# Connect to Netezza database\nconn = nzsql.connect('\'localhost\'', '\"myusername\"', '\"mypassword\"', '\"mydatabase\"')\n\n# Create a cursor object\nncursor = conn.cursor()\n\n# Define the SQL statement\nsql = \"\"\"\nSELECT *\nFROM CUSTOMER\nJOIN TRANSACTIONS\nON CUSTOMER.CUSTOMER_ID = TRANSACTIONS.CUSTOMER_ID\n\"\"\n\n# Execute the SQL statement\nncursor.execute(sql)\n\n# Fetch the results\nresults = ncursor.fetchall()\n\n# Print the results\nfor row in results:\n print(row)\n\n# Close the cursor and connection\nncursor.close()\nconn.close()\n```\n\nThis Python code uses the nzsql library to connect to a Netezza database, create a cursor object, define the SQL statement, execute the statement, fetch the results, and print the results. Finally, it closes the cursor and connection.\n[INST]",
 "parameters": {
 "decoding_method": "greedy",
 "max_new_tokens": 900,
 "min_new_tokens": 0,
 "stop_sequences": [],
 "repetition_penalty": 1
 },
 "model_id": "meta-llama/llama-2-70b-chat".
}
```

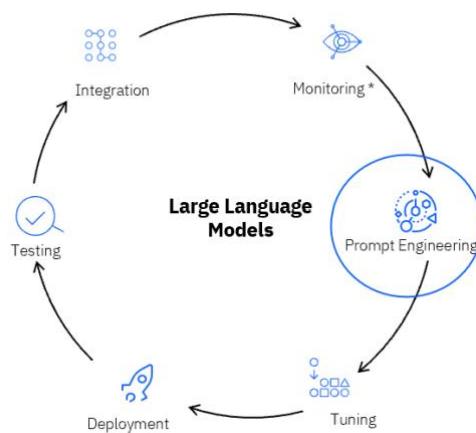
Another interesting part of the prompt are system instructions: to return output in markdown formatting vs. just the default text. When output is returned in this format, the integration with the UI (in this case the Chat UI interface) requires fewer lines of code.

Watsonx.ai does not provide a UI for end-user applications because UI is typically developed and hosted in other components of a solution. In the next section you will learn how to integrate LLMs with applications.

## **TODO: Final note of model selection – explain why they should consider granite**

## Part 2: Integrate LLMs with applications

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.



Let's review other steps:

- There are several types of tuning, which is usually applied to improve model output. In many use cases tuning won't be a required step.
- LLMs are pre-deployed (available for invocation out-of-the-box) in *watsonx.ai*. The only time deployment may be required is for tuned models.
- Testing and integration is done with either the REST API or the Python SDK.

In this section we will review the testing and integration steps.

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 *granite-13b-chat-v2* or the *llama* model (add the system prompt for llama)
- 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.

Chat Structured Freeform AI Model: granite-13b-chat-v2 {#} </> X

payment.

4. These discrepancies in the cash payout amount and the reduced mortgage payment relief prevented the reviewer from fulfilling their original financial objective of obtaining the \$25K cash.

5. The limitations and changes in the terms impacted the overall satisfaction with the refinancing process and compromised the financial benefits the reviewer had anticipated.

Review:""

I started my loan process toward securing a VA loan. I was waiting for a month and a couple weeks, then I was told that the VA needed to acquire my retirement points to verify my veteran status. If I knew this is what my loan was on hold for, I could have contacted the VA office right away and got this cleared up.

For whatever reason, it took the underwriting department a long time to verify my employment status, even after I uploaded a couple years of my W2 forms from both of my jobs, and they had my Social Security number to further verify my employment status. My loan completion date was extended, because I wasn't made aware that they were waiting for my VA status to be approved. The push back for my mortgage is common for mortgage companies, but this caused my interest rate to go up. Then, the securing of a closing lawyer being made aware to me and the lawyer needing three days to get their end prepared for me to go to their office to sign the paperwork wasn't made aware to me. My loan missed the second closing date. For whatever reason, the locked in interest rate jumped up 5/8 points. After making the banker I was working with aware of this, he didn't understand why the locked in interest rate jumped up either. He was nice enough to work on it and was able to get the interest rate down in 1/4 of a point, so my mortgage has an interest rate that is 3/8 of a point higher than my locked in interest rate in the beginning of this process. Although my interest rate is higher than the locked in interest point, at the end, the mortgage is successfully finished.

""

Top bullet points:1. The reviewer started the process for a VA loan and experienced delays due to the need to verify their veteran status and employment status.

2. The loan completion date was extended due to these verification issues, resulting in a higher interest rate.

**Model parameters**

Decoding

Greedy  Sampling ⓘ

Repetition penalty

1 ⏹ 2 1

Stopping criteria ⓘ

Stop sequences

. × +

Min tokens 50 Max tokens 300

Enter up to 6 sequences to stop output after the minimum number of tokens is reached.

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

AI Model: granite-13b-chat-v2 {#} </> ⌂

Copy the code to a notepad.

[View code](#)[Curl](#) [Node.js](#) [Python](#)

```
curl "https://us-south.ml.cloud.ibm.com/ml/v1/text/generation?version=2023-05-29" \
-H 'Content-Type: application/json' \
-H 'Accept: application/json' \
-H 'Authorization: Bearer YOUR_ACCESS_TOKEN' \
-d '{
 "input": "Please provide top 5 bullet points in the review provided in ''''''''.
\n\n\nReview:\n''''''''I had 2 problems with my experience with my refinance. 1) The appraisal company used only tried to lower my house value to fit the comps that he was able to find in the area. My house is unique and he did not use the unique pictures to compare value. He purposely left them out of the appraisal."}
```

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.

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

Header

"input": "Please provide top 5 bullet points in the review provided in \n\nReview:\n\nI had 2 problems with my experience with my refinance. 1) The appraisal company used only tried to lower my house value to fit the comps that he was able to find in the area. My house is unique and he did not use the unique pictures to compare value. He purposely left them out of the appraisal. 2) I started my loan process on a Thursday. On Saturday I tried to contact my loan officer to tell him of the American Express offer that I wanted to apply for. I was informed that it was too late and I could not use it because it would delay the process. I had just received the email about the offer and I had just started the process so how was it too late to get in on the \$2,000 credit on my current bill. I let it go but I should have dropped the process and restarted it because that would have helped me out with my bill.'''''\\nTop bullet points:\\n1. The appraisal company undervalued the reviewer''s house by purposely excluding unique pictures that would have accurately assessed its value.\\n2. The uniqueness of the house was not taken into consideration, and the appraiser relied solely on comps that did not reflect its true worth.\\n3. The reviewer attempted to inform their loan officer about an American Express offer they wanted to apply for, which would have provided a \$2,000 credit on their current bill.\\n4. The loan officer stated it was too late to take advantage of the offer as it would delay the process, despite the reviewer having just received the email and recently started the loan process.\\n5. The reviewer regrets not

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
},
"model_id": "ibm/granite-13b-chat-v2",
"project_id": "2e0ffc21-9d10-4cde-9fce-f3e78f7503e3",
```

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

Projects / LLM-Workshop-YI

Overview Assets Jobs Manage

Project

**General**

- Access control
- Environments
- Resource usage
- Services & integrations

General

**Details**

Name  
LLM-Workshop-YI

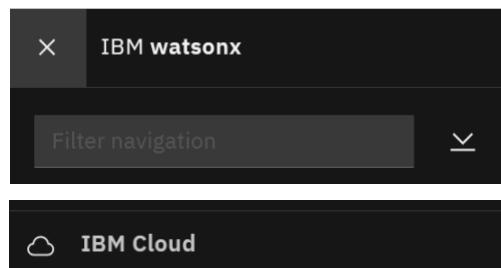
Description  
What's the purpose of this project?

Project ID  
e9207503-90c4-430b-a366-d6c24b5a020a

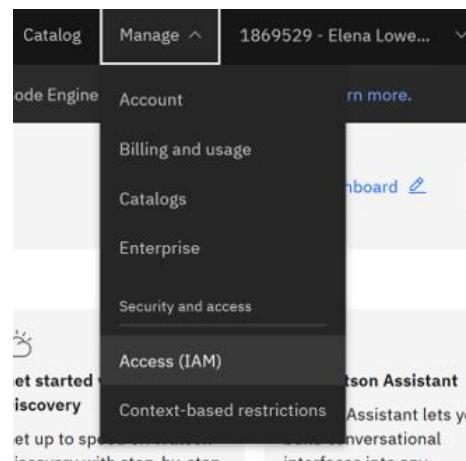
Now we will get the authentication token.



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.

The screenshot shows the 'API keys' page under the 'Access (IAM)' section. On the left, there's a sidebar with options like 'Manage identities', 'Users', 'Trusted profiles', 'Service IDs', 'API keys' (which is selected and highlighted in blue), 'Identity providers', 'Manage access', 'Access groups', 'Authorizations', 'Roles', 'Gain insight', and 'Inactive identities'. The main content area has a heading 'API keys' and a paragraph about managing API keys. It includes a link to 'IBM Cloud® Secrets Manager' for creating and leasing API keys dynamically. Below this is a table with a single row:

| Status | Name                    | Description | Date Created         |
|--------|-------------------------|-------------|----------------------|
|        | API Key for Deployments |             | 2021-12-13 15:23 GMT |

A blue 'Create +' button is located at the top right of the table.

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

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



Click **Create** to import the notebook.

Work with data and models in Python or R notebooks

Define the details to create a notebook asset and open it in the Jupyter notebook editor tool.

Blank      **From file**      From URL

Name  
TestLLM

Description (optional)

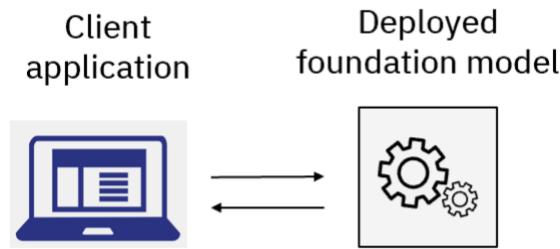
Select runtime  
Runtime 22.2 on Python 3.10 XXS (1 vCPU 4 GB RAM)

The selected runtime has 1 vCPU and 4 GB RAM.  
It consumes 0.5 capacity units per hour.  
[Learn more](#) about capacity unit hours and Watson Studio pricing plans.

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.

**Note: if you do not have a Python IDE, skip the rest of the lab and watch this video in the lab repo/Reference to learn how to invoke LLMs from client applications. Proceed to the Conclusion section.**

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

29. Find the following Python scripts in the downloaded *lab repo/Scripts* folder:

- *demo\_wml\_api.py*
- *demo\_wml\_api\_with\_streamlit.py*

Load these scripts into your Python IDE.

30. In your Python IDE install the required libraries. We recommend that you use Python 3.10 environment but 3.11 can be also used.

```
pip install ibm-watsonx-ai
```



```
pip install ibm-cloud-sdk-core
```

In non-Anaconda Python environments you may need to install another package:

```
pip install python-dotenv
```

Let's review the scripts.

*demo\_wml\_api.py* is a simple Python script that shows how to 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\_credentials()*: reads the api key and the project id from the .env file (will be used for authentication)
- *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, create a .env file in the root directory of your project and add your *Cloud API key*, *project id*, *cloud URL*, and *locale*.

The template for the .env file is located in the lab *repo/Reference* folder (*env\_file\_template.txt*)

```
Project .env use_case_generate.py use_case_inference.py
LLM_Workshop C:\Users\1A3030897\PycharmProjects\LLM_Workshop
 1 api_key=YgIJBtEHsjJ_P1AC8bhW7loXzTDd
 2 project_id=ca2e06c0-c0dc-4114-8ec5-4
 3 url=https://jp-tok.ml.cloud.ibm.com
 4 # use jp for Japan and us for US
 5 locale=jp
```

31. 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 *lab repo/Reference* folder.

32. 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. If you're not able to install Anaconda in your environment, please check with your instructor.

33. 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 4 functions:

- *get\_credentials()*: reads the api key and the project id from the .env file (will be used for authentication)
- *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.

34. 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.