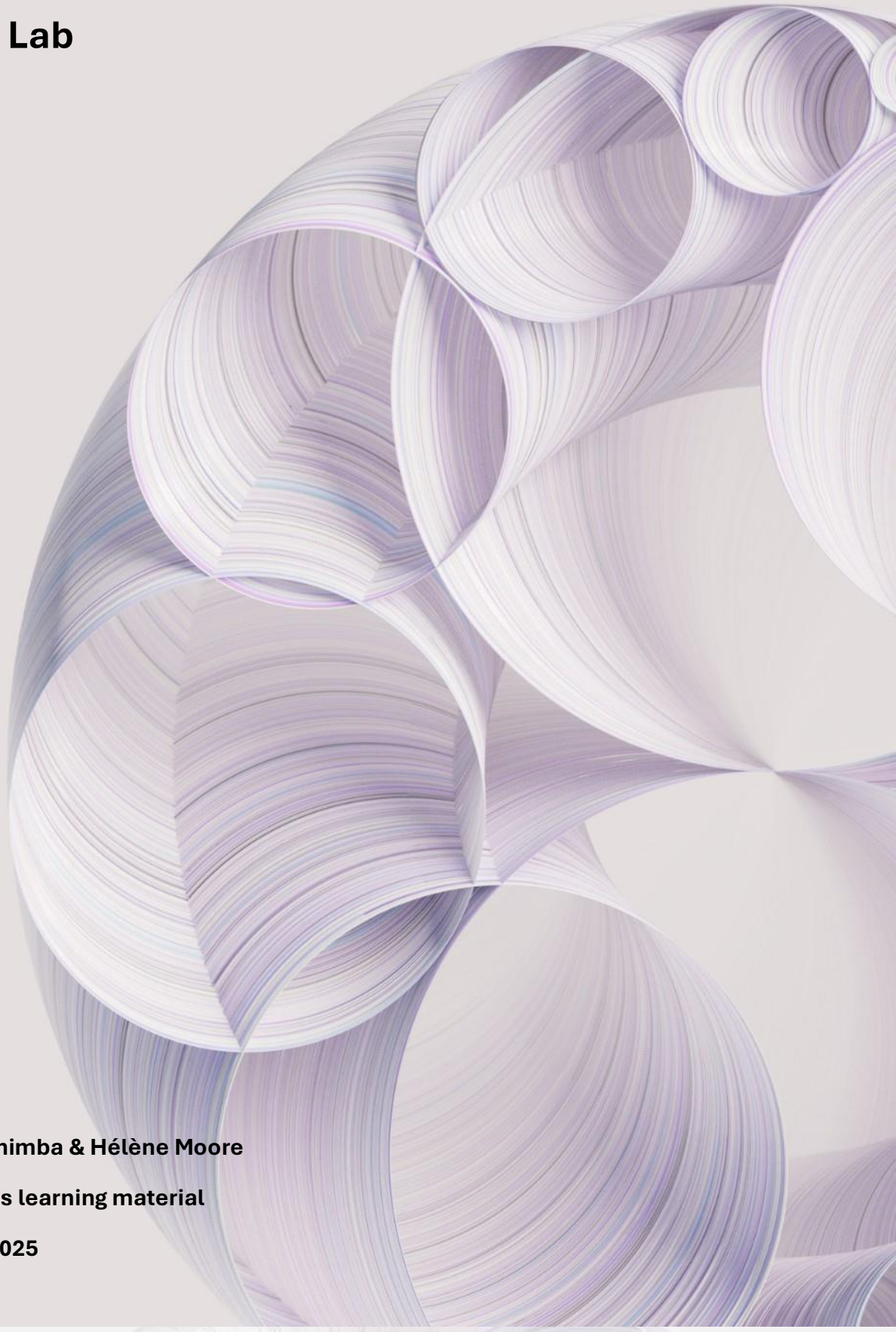
IBM watsonx Gen AI Challenge

Lab 1

IBM Prompt Lab



Authors: Florian Kanimba & Hélène Moore

Based on: Felix Lee's learning material

Last edited: 24.02.2025

Zurich, CH

Acknowledgment

We would like to thank Felix Lee, Charleen Englert and Nicolas Kohler, as this Lab uses learning material they created and updated. This includes Task 2 as well as all the following tasks. Only minor modifications have been made to their content, such as small updates to the content, formatting etc.

Disclaimer

Please note that **some of the outputs that you will generate will not match the outputs discussed in this lab.** Your goal is not to recreate the outputs line by line but to get a general feel and understanding about what the Prompt Lab has to offer and how changing different settings and parameters can influence the results.

The reason for this is because watsonx.ai is being developed and released in an agile manner. In addition to the above, the web interface is likely to change over time. Therefore, the screenshots used in this lab may not always look exactly like what you see. You can expect to encounter some of the following:

- Additional foundation models in the library list.
- Changes in the user interface (location of buttons, text for various fields)
- Additional tabs/buttons (especially when the Tuning Studio is rolled out)

These should not affect how the labs work but have patience and explore.

There are three changes, however, that can affect the results

- Foundation models can be very sensitive to input. If you enter slightly different text than what the exercise is using (even if it is just one single word), the outcome can be different.
- **There is ongoing tuning of the models. If the models themselves are updated, then some of the results may vary.**
- The sample prompts may change (more can be added, or updated with new text). Since prompt examples are used heavily in this lab, this can affect what you might see. To mitigate that, the prompt example text used will be provided in a gray text box. If you select a prompt example and the text is different, simply copy the given text into various input fields.

Content

This IBM watsonx.ai hands-on lab introduces you to some of the core components and capabilities of IBM watsonx.ai. Specifically, you will get hands-on experience in the following areas:

- Strengths and weaknesses of different models
- Meaning of different configuration parameters and how they influence a model in generating output
- Zero-shot and one-shot prompting (and by extension few-shot prompting)
- Generating list and JSON files with foundation models
- Code generation and translation
- Saving prompts as well as prompt sessions (to keep the history of all your changes)
- Restoring the prompt to an earlier state via prompt history
- Saving prompts to a Jupyter notebook and working with the Jupyter notebook

To understand more about the interface of watsonx.ai, we highly encourage you to read the document ***'How to get to watsonx'*** where the web-based user interface (UI) is described more thoroughly. It includes the Prompt Lab, the Structured and Freeform interface, model parameter configuration panels, and model information panels.

Summary

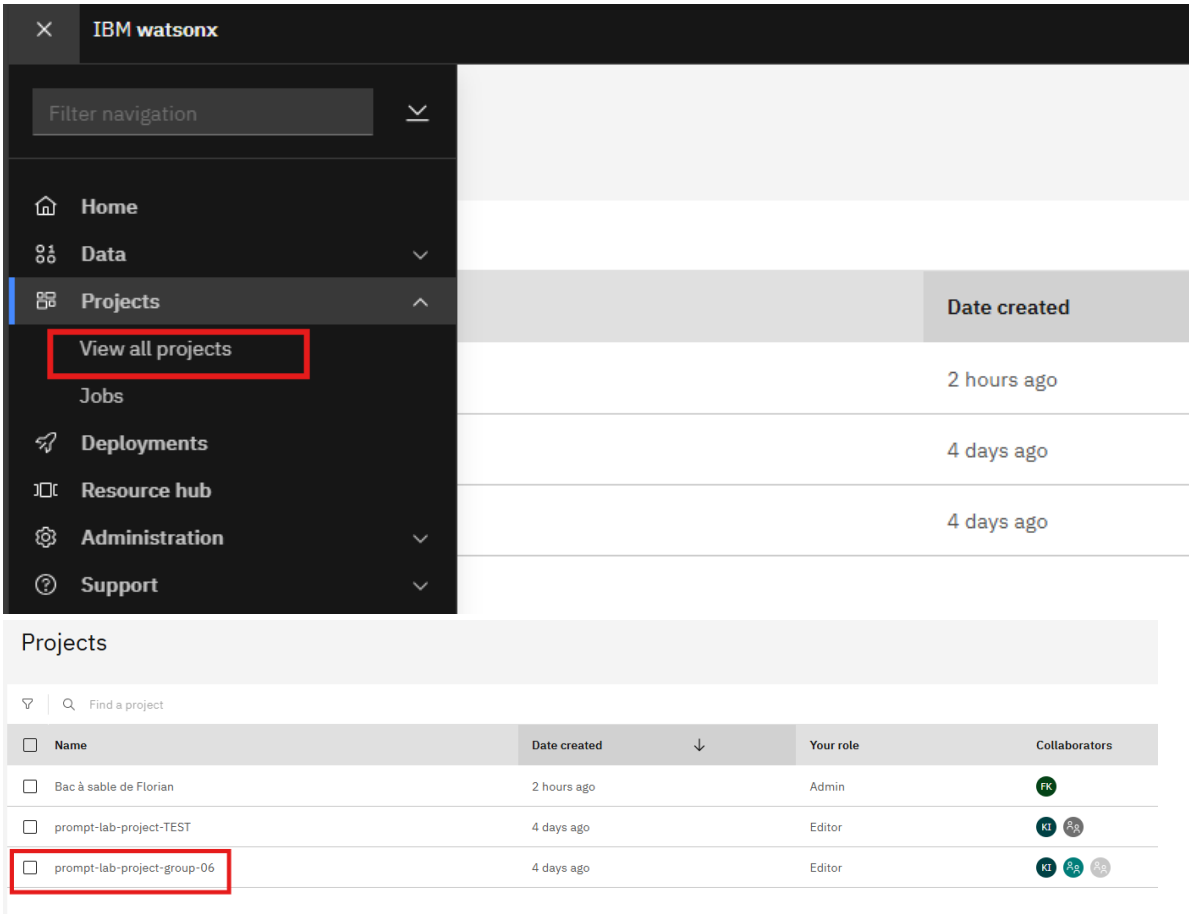
At the end of this lab, you will be able to open a new Prompt Lab session and navigate the watsonx.ai interface, including selecting and experimenting with different foundation models. You will understand the importance of prompt engineering and how to modify prompts familiar with zero-shot, one-shot, and few-shot prompting techniques and how to use them effectively. Additionally, you will learn to adjust inference parameters such as temperature, top P, and top K to influence model outputs. You will also be able to generate structured outputs like lists and JSON, and use stop sequences to control model generation. Finally, you will know how to save prompts and prompt sessions, and understand the importance of testing model outputs, especially for code-based tasks.

Contents

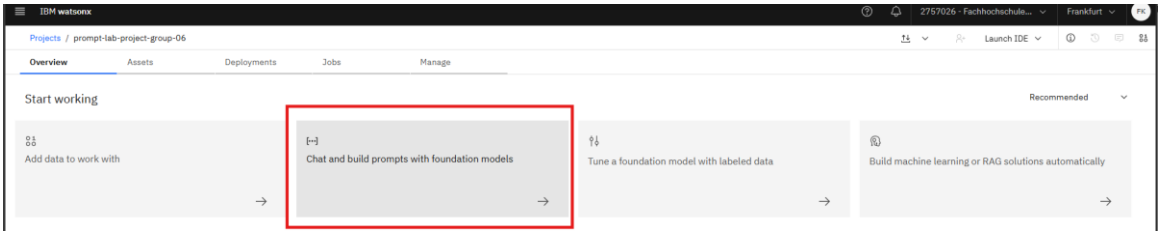
Prompt Lab

1. Go to your project and open Prompt Lab

1. From the Watsonx.ai home, access the **Projects** section from the left menu. Select **"All Projects"**, where you should see your assigned project, labeled from **01 to 22**.



2. Navigate to **Prompt Lab**.

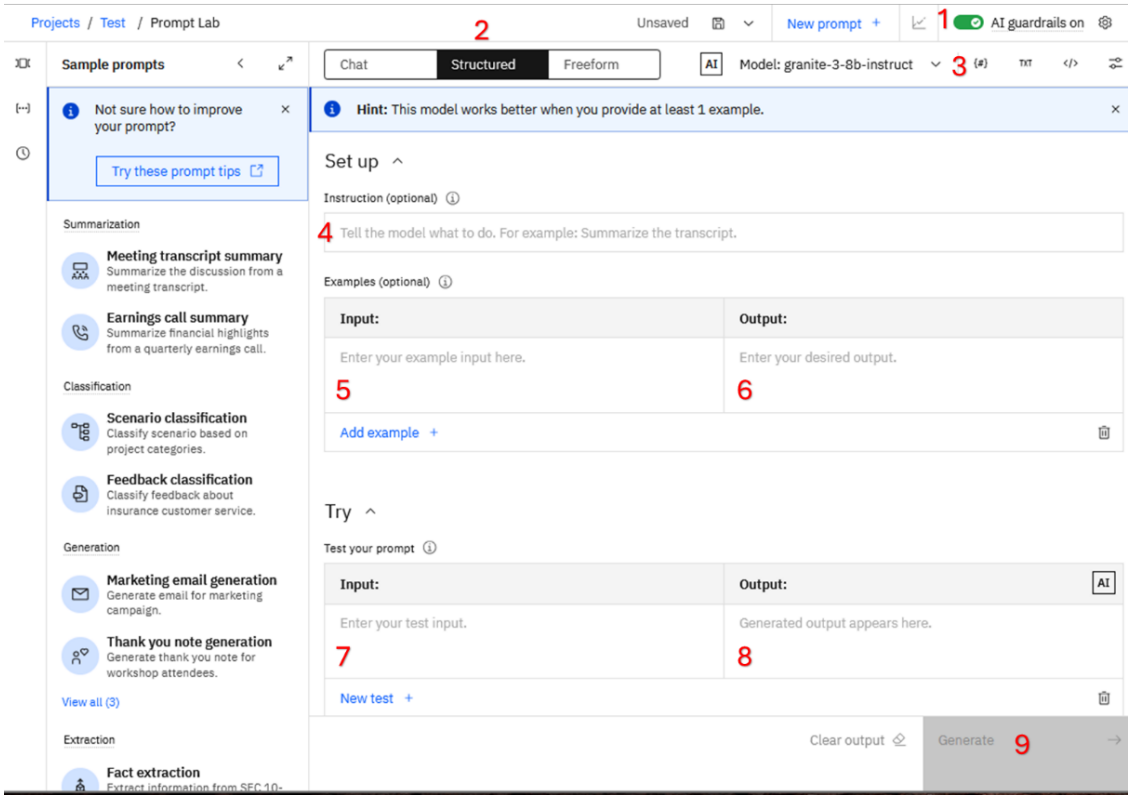


2. Understanding and choosing the right Foundation Models

Foundation models are large-scale pre-trained models that serve as the base for various AI applications. All supported foundation models in watsonx.ai can be found here.

2.1 Prompt lab mods [Chat, Stuctured, Freeform]

Prompt Lab console



1. **AI guardrails**

   Watsonx.ai provides AI guardrails. You turn this on to prevent potential harmful input and output text (such as hate, abusive, or prejudiced wordings). You might have to start working with watsonx.ai before you can turn this on.

2. **Mods**

   Start exploring the different mods, using the Sample prompts

   Chat:

The "Chat" mod is tailored for creating and managing conversational AI interactions.
It allows you to simulate dialogues, test responses, and refine the conversational flow.
You can input prompts and receive responses in real-time, making it ideal for developing chatbots or virtual assistants.

Structure:
The "Structure" menu is designed for handling structured data inputs.
It provides templates and predefined formats to ensure data consistency.
This menu is useful for tasks that require specific data formats, such as filling out forms or working with tabular data.

Freeform:
The "Freeform" menu offers a flexible input environment where you can provide data without any constraints.
It is suitable for, creative tasks, brainstorming, or input unstructured data
This menu allows for a more open-ended interaction with the AI, enabling you to explore various possibilities and use cases.

3. **Model selection**

You can select the model best suited to your task, for each menu. Select “Select all foundation models” and use the benchmark table to help you make the right chose. **Warning – make sure to select models that are not on demand, to avoid generate costs**

2.2 Prompt lab mods [Structured] specificities

4. **Instructions [Structured]**

Allows users to provide an instruction which will become part of your prompt.
While it is optional, you should be careful what you place here because it consumes tokens.
Keep in mind that prompting is not the same as asking a question. A foundation model is not “answering” but rather generating output based on the input.
Anything that is included can be used by the model.

5-6. **Examples: Input – Output [Structured]**

Foundation models can sometimes be thought of as probability machines – they generate output based on what might be the best extension of the given input (prompt). However, unless a user is an expert in writing prompts, a foundation

model may not understand the intent of the prompt, or the type/structure desired for the output.
A user can “teach” a foundation model by providing a sample input and output (referred to as a “shot”). These instruct the model on how to best respond to a query/prompt. Types of shots include:

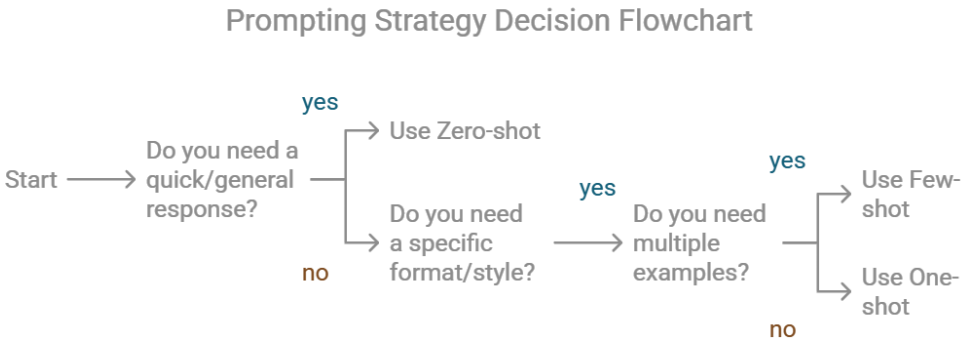
Zero-shot prompting (0 example)	**One-shot prompting** (1 example)	**Few-shot prompting** (multiple examples)
No example data is provided. The model responds based solely on its pre-trained knowledge.	A single input/output example is given to help the model understand the expected response format.	Several examples are provided to help the model recognize patterns and generate more precise responses
☑ Quick and easy to implement.	☑ Slightly guides the model toward a specific format.	☑ Higher accuracy and better adaptation to specific needs.
✕ Less accurate for specific tasks.	✕ May not be sufficient for complex tasks.	✕ Requires more preparation and consumes more resources.

Zero-shot Prompting – General Response
If you don’t provide example input and output, the model generates an output based purely on its pre-trained knowledge, without any specific examples.

One-shot Prompting – Guided Structure
A single example is provided to shape the response’s format and tone.
Example input provided: "Write a book summary for ‘Harry Potter and the Philosopher’s Stone’ "

Example output provided: “ ‘Harry Potter and the Philosopher's Stone' is a delightful fantasy novel that introduces us to the magical world of Hogwarts and its young protagonist, Harry Potter"

Few-shot Prompting – Pattern Recognition
Multiple examples are given to help the model recognize patterns and variations.



Notes:

- Prompts are tokenized before being passed into a model, and typically foundation model usage costs are calculated based on the number of tokens. Using few shots prompting frequently can become expensive.

- As a related concept, the larger the foundation model the more resources (cores and memory) and costs are required to operate them.

- Keep in mind that model parameters or tuning can have a heavier impact on the output.

7-8-9. Try: Input – Output [Structured]

The Try section is where you enter your prompt/query into the foundation model. Click the Generate button for the model to generate the output. The output from a foundation model will be displayed.

3. Setting Model Parameters

For each mods (Chat, Structured, Freeform), you have access to the model parameters. Model parameters can adjust the model to the example following:

- *Temperature* - Degree of freedom which the model will answer your prompt
- *Stopping criteria* - control when to stop generating output
- *Top P (nucleus sampling)* dynamically selects the smallest set of next-word candidates whose cumulative probability exceeds **P**, allowing more randomness and creativity as **P** increases.

4. Tuning (to go further)

Tuning refers to optimizing hyperparameters, which are settings that define how the model learns. Unlike parameters, these are manually set before training and can significantly impact model performance.
Tuning can help your prompt achieve more precise **labeling**, **generation**, and **summarization** as regular prompt engineering.

You can for the moment tune two foundation Models in Watson X.

- granite-13b-instruct-v2
- flan-t5-xl-3b

When to Tune a Model?	*When not to Tune a Model?*
• Get the model's output to use a certain style or format • Improve the model's performance by teaching the model a specialized task • Generate output in a reliable form in response to zero-shot prompts	• Improve the accuracy of answers in model output • Get the model to use a specific vocabulary in its output consistently • Teach a foundation model to perform an entirely new task

Example:
As an insurance company, you have loads of customer claims in unstructured data (email, notes...) It takes lot of manual effort to see each claim and label it and redirect it to the right department. In our example you can then label your unstructured data into company-specific categories, summarize the claim, and guide employee for next action to take. To do this, you provide a few input and output examples in the **Structured** menu to guide the model's learning and improve its accuracy.

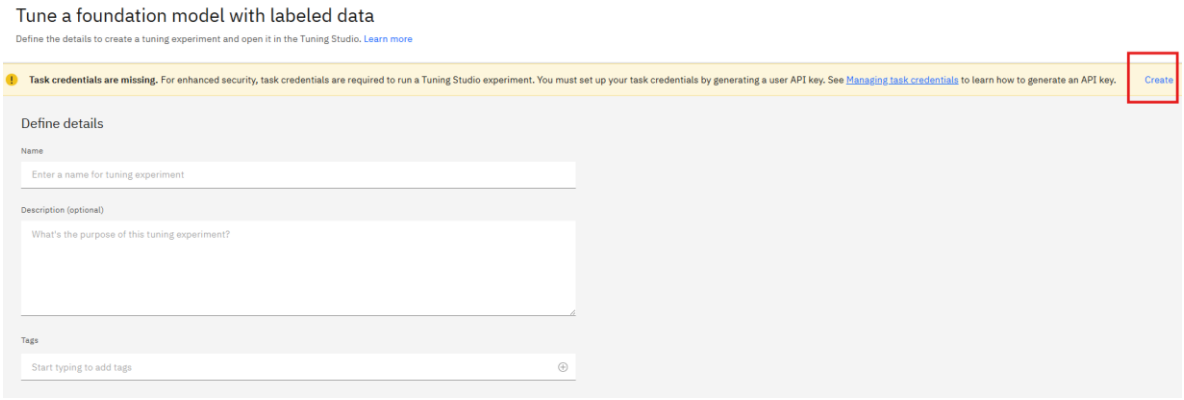
	Input	**Output**
Example 1 training data provided	*"I filed a claim for my medical expenses weeks ago, but the insurance company keeps delaying payment with no clear explanation."*	**Category:** Claim Processing **Sub-product:** Health Insurance **Issue:** Delayed Payment Customer is experiencing unexplained delays in claim reimbursement.

		Review claim status and provide a clear update or expedite processing.
Example 2 training data provided	*"My car was damaged in an accident, and my policy covers repairs, but the insurance company denied my claim without a valid reason."*	**Category:** Claim Denial **Sub-product:** Auto Insurance **Issue:** Unjustified Claim Rejection Claim for vehicle damage was denied despite policy coverage. Reassess claim and communicate the exact reason for denial or reconsider based on policy terms.
Result (output generated by the model)	*"My medical claim was denied, even though my doctor confirmed the treatment was necessary. Now I'm stuck with the full bill and no support from my insurer."*	**Category:** Claim Denial **Sub-product:** Health Insurance **Issue:** Unjustified Claim Rejection A necessary medical treatment claim was denied, leaving the customer with full costs. Re-evaluate claim based on medical necessity and policy coverage; provide an appeal process if applicable.

4.1 How to tune your Model

This video shows you how to tune a foundation model. There are a couple of reasons why you may want to tune your foundation model. By tuning a model on a large number of labelled examples, you can enhance the model performance compared to prompt engineering alone. Or by tuning a base model to perform similarly to a significantly bigger model, you can reduce costs by deploying that smaller model.
In addition to the video, check also this tutorial to discover more on how to tune a foundation model.

Note:
When accessing the Tuning Lab, you may encounter a warning message related to the API. Simply click "**Create**", then click "**Create key**". The Tuning Lab will then be accessible.



Bonus Tasks

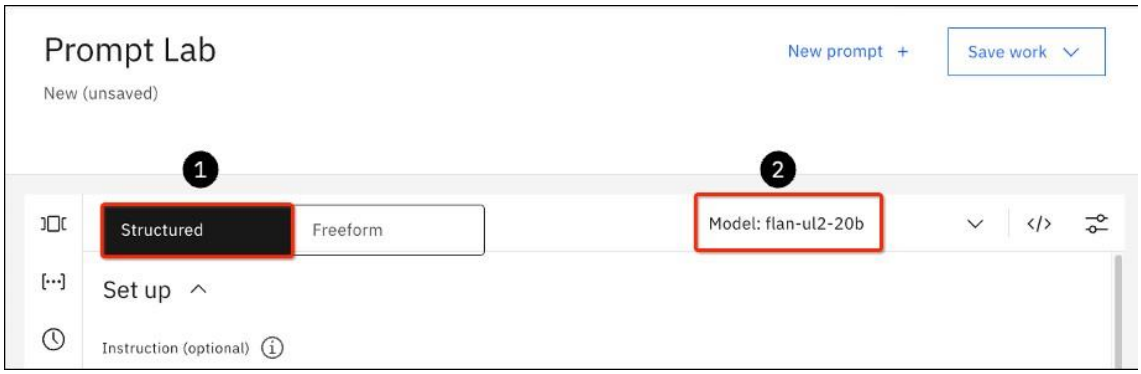
For this bonus part, you will explore additional capabilities of the watsonx.ai platform, specifically working with another data format (JSON) and creating a Jupyter notebook from a prompt. These tasks are designed to help you discover more about the platform's versatility and functionality.

1. Exploring with another data format (JSON)

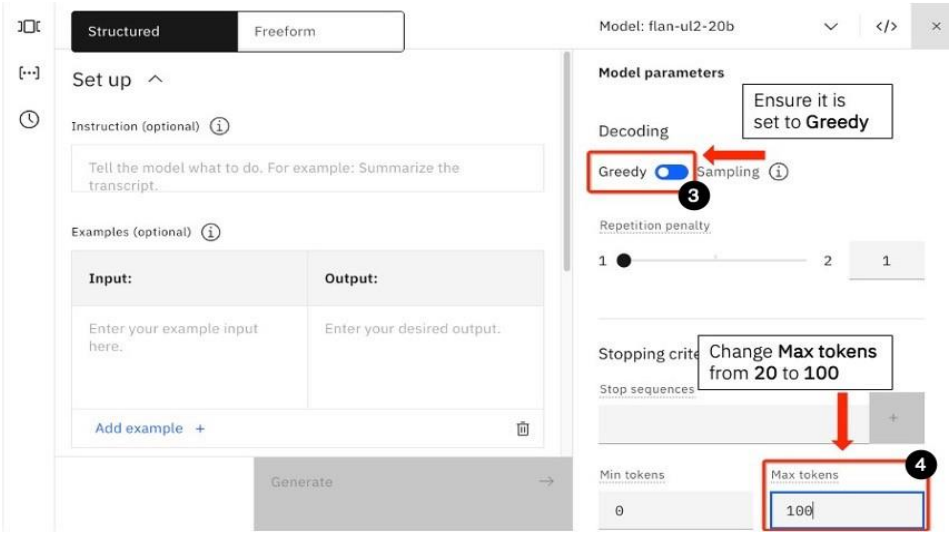
In the JSON task, you will use the Prompt Lab to generate a simple JSON file, experimenting with different models to see how they handle structured output. You will learn that while some models excel in generating natural language text, others, like the mpt-7b-instruct2 and starcoder-15.5b, are better suited for producing valid JSON outputs. Additionally, you will explore the use of Stop sequences to control the model's output and prevent unwanted text generation.

In this section, you will use the Prompt Lab to generate a simple JSON file.

1. Open a new Prompt Lab session in the Structured mode.
2. Ensure that you are using the flan-ul2-20b model.



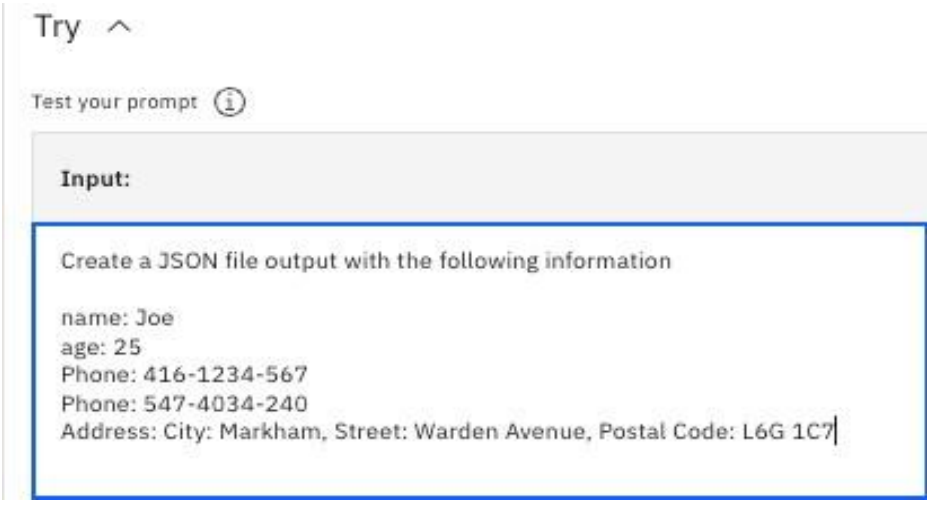
3. Ensure that you are using the Greedy mode.
4. Set Max tokens to 100.



5. Copy and paste the following to the Input field under the Try section

> Create a JSON file output with the following information
>
> name: Joe age: 25
>
> Phone: 416-1234-567
>
> Phone: 547-4034-240
>
> Address: City: Markham, Street: Warden Avenue, Postal Code: L6G 1C7

Your input field should now look like this:



Click Generate. The flan-ul2-20b model returns the following output:

Output:

Joe, age 25, has two phone numbers, 416-1234-567 and 547-4034-240. His address is in Markham, Warden Avenue, L6G 1C7.

This shows the strengths of the flan-ul2-20b model in that it responded with a natural language output. However, it clearly is not the JSON output you are looking for.

6. Switch to the flan-t5-xxl-11b model and again click Generate. Similar to the flan-ul2- 20b model, you will see this output:

Output:

Joe is 25 years old and lives at Warden Avenue, Markham, Ontario, Canada. He can be reached at 416-1234-567 and 547-4034-240.

Again the completion provides a good natural language output, but it is not in JSON. At this juncture, you can try using one-shot or few-shot prompting to help the model understand how you want to structure the output. However, this would not be an easy task. Instead, we will move on and try other models.

7. Now try the mpt-7b-instruct2 model and click Generate. You will see this output:

Output:

```
{
 "name": "Joe",
 "age": 25,
 "Phone": [
  "416-1234-567",
  "547-4034-240"
 ],
 "Address": {
  "City": "Markham",
  "Street": "Warden Avenue",
  "Postal Code": "L6G 1C7"
 }
}
```

This is a valid JSON output for the given input. The mpt-7b-instruct2 has shown that it understands how to translate input text to JSON.

8. Now try another model that we have not yet experimented with. Select the starcoder-15.5b model and then click on Generate. You get the following output.

Output:

```
{
  "name": "Joe",
  "age": 25,
  "phone": [
    "416-1234-567",
    "547-4034-240"
  ],
  "address": {
    "city": "Markham",
    "street": "Warden Avenue",
    "postal_code": "L6G 1C7"
  }
}

"""
```

The starcoder-15.5b model also generates a valid JSON output, except that it has triple double quotes (“”””) at the bottom.

9. While this is not a big issue, we will now use the Stop sequences configuration setting to eliminate the triple double quotes.

   In this particular case, you want to stop the generation once the model encounters the following text sequence (2 curly brackets that conclude the JSON section).

   }

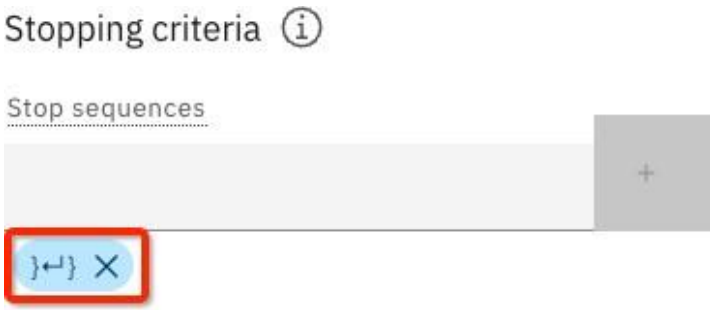
   }

   In the upper right corner of the console, click the icon to slide open the configuration parameters. Enter the following keystrokes on the Stop sequences field:

   } <carriage return> }

10. You should see the following, then click on the blue + button to add the sequence.

The stop sequence is now added. You will see this:

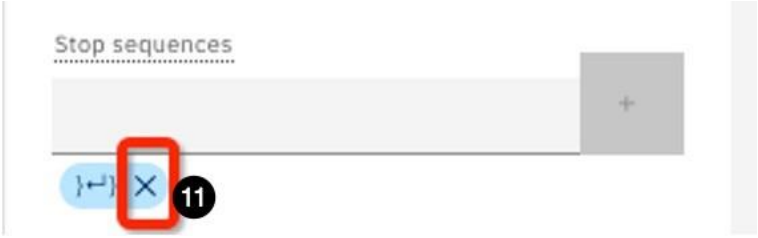


Now click Generate. The output (see below) no longer includes the triple double quotes as the model recognizes the stop sequence and ceases to generate anything more.

Output:


```
{
  "name": "Joe",
  "age": 25,
  "phone": [
    "416-1234-567",
    "547-4034-240"
  ],
  "address": {
    "city": "Markham",
    "street": "Warden Avenue",
    "postal_code": "L6G 1C7"
  }
}
```


11. You can remove the Stop sequence by clicking X.



Bonus task/questions:

- Remove any Stop sequences. What if you specify just one curly right bracket instead? Will it work? Try it out, does it do what you expected?
- Again remove the previous Stop sequence. What if you specify 2 curly right brackets (with no carriage return in between) like this }}, will it work? Try it out, does it do what you expected?

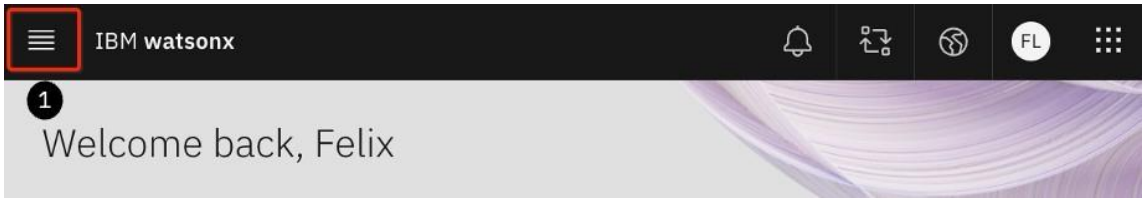
2. Using Jupyter notebooks with prompts

You have been working with prompt engineering via the console. However, this is not how data and AI engineers typically work. In this section, you will create a Jupyter notebook for a prompt, and work with prompts in a Jupyter notebook in watsonx.ai.

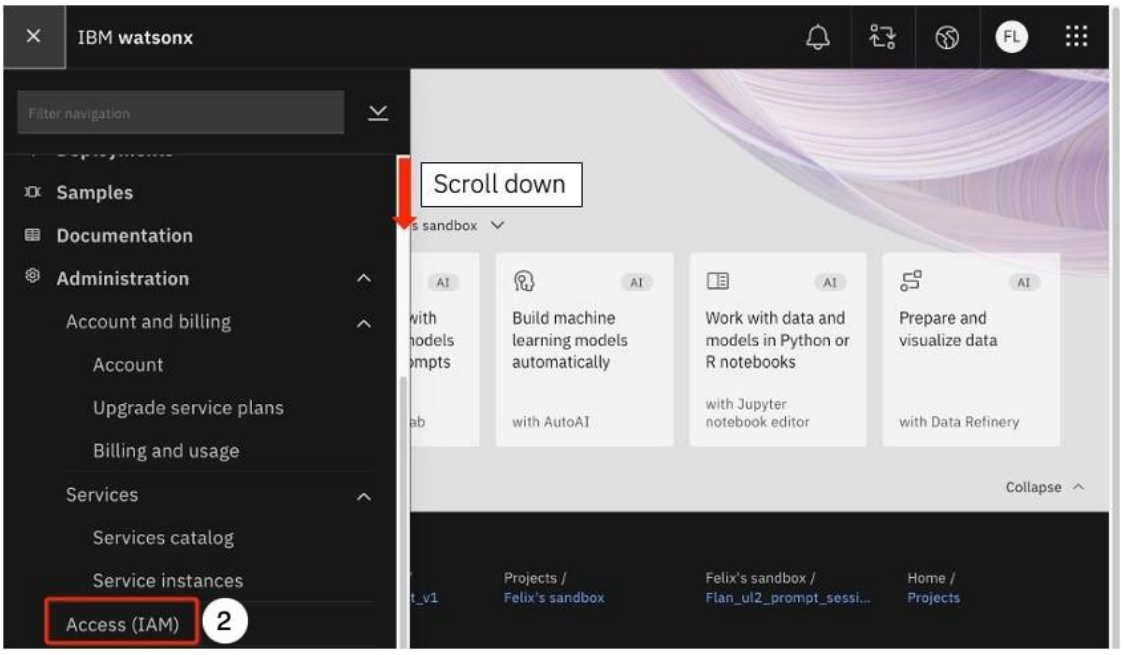
a. Getting your API key

To run a Jupyter notebook you will need your API key. If you have not generated one before, you can do so by following the steps below.

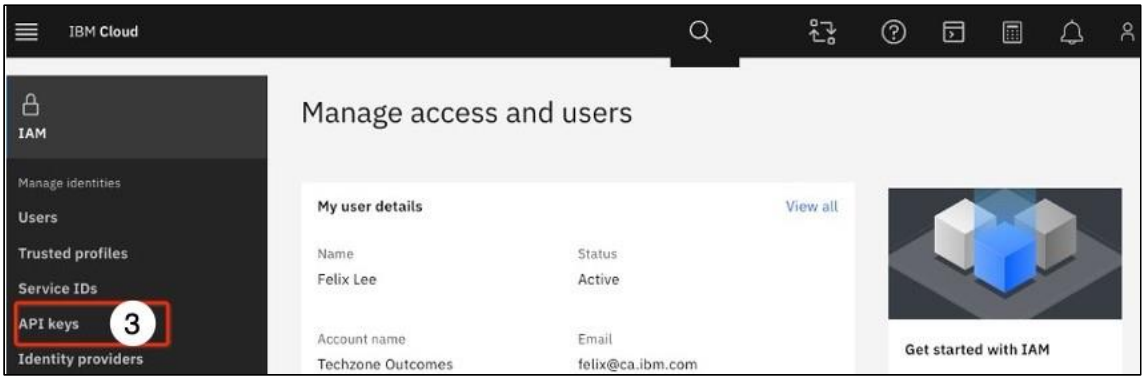
1. Log onto the IBM Cloud Console, or if you are using the Prompt Lab, you can click the icon in the upper left corner.



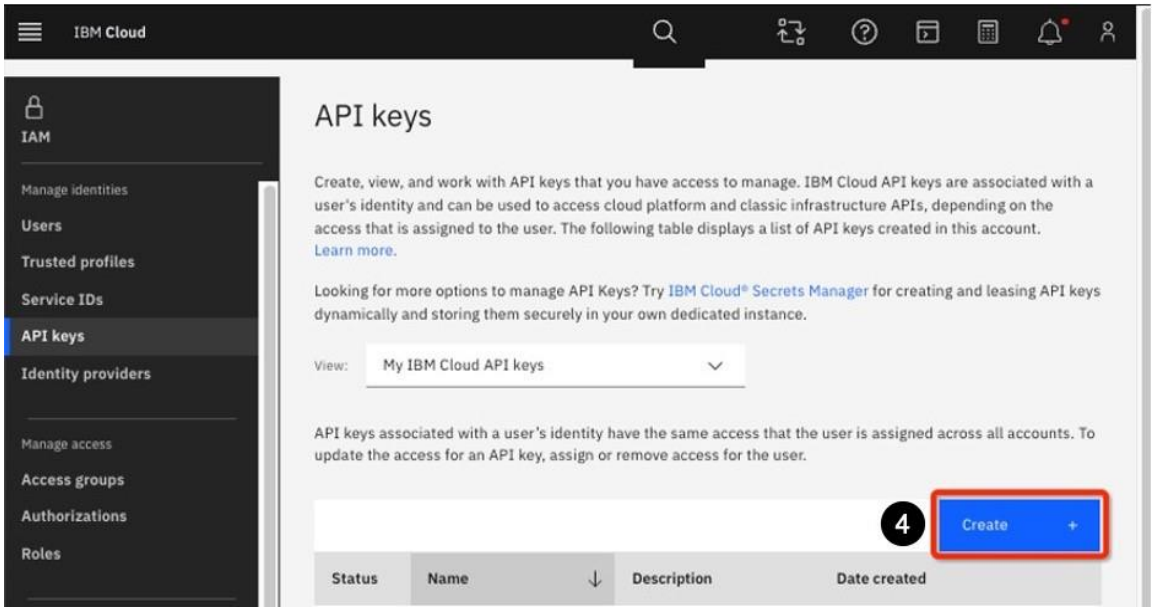
2. Scroll down on the slide out panel and click the option Access (IAM) option from the pop-up menu (you may have to log into the cloud console).



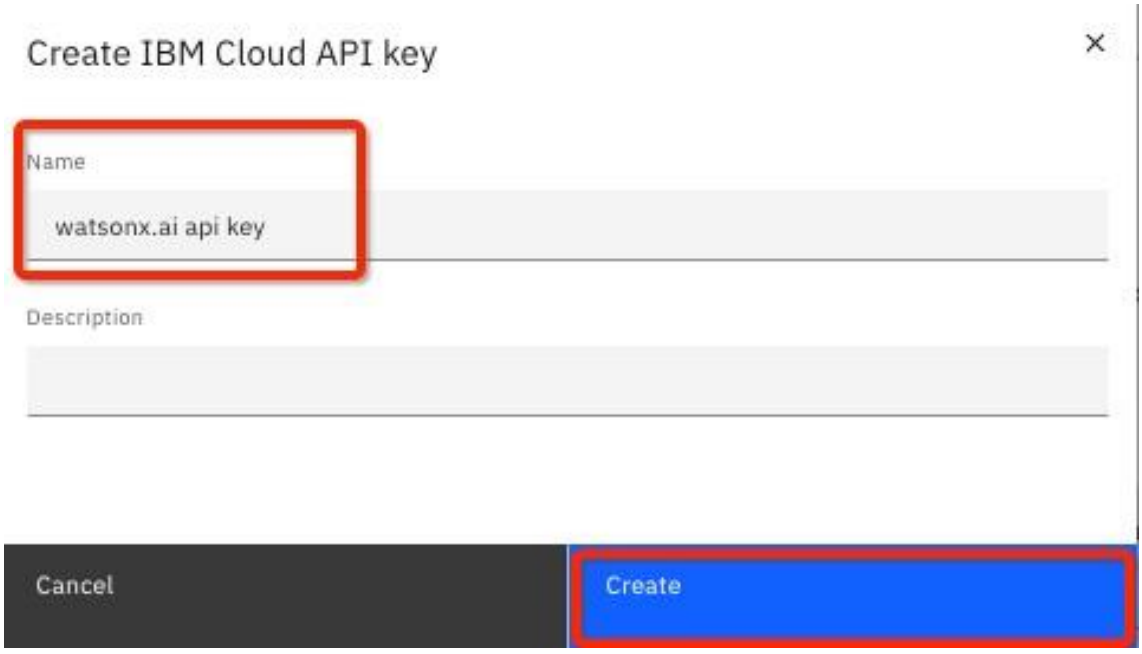
3. Select Access (IAM) to bring up the Manage access and users panel. Find and select the API keys item from the left-hand panel.



4. On the API keys panel, click Create.

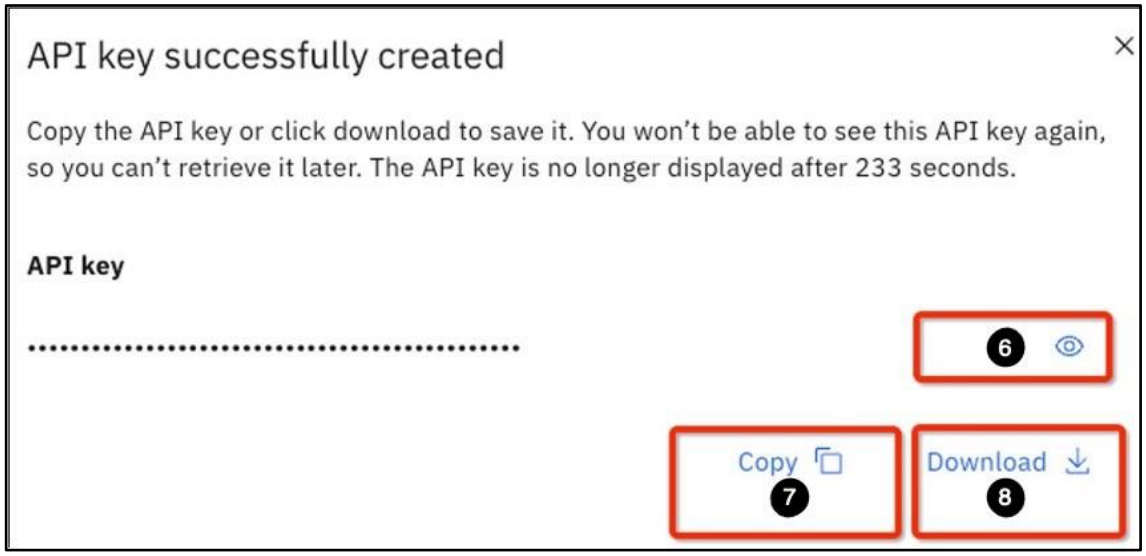


5. On the next panel – you need to provide a name. You can use any name. In the example here, the name watsonx.ai api key is used. Now click Create.



An API key will be created for you. On the resulting panel, you have several choices:

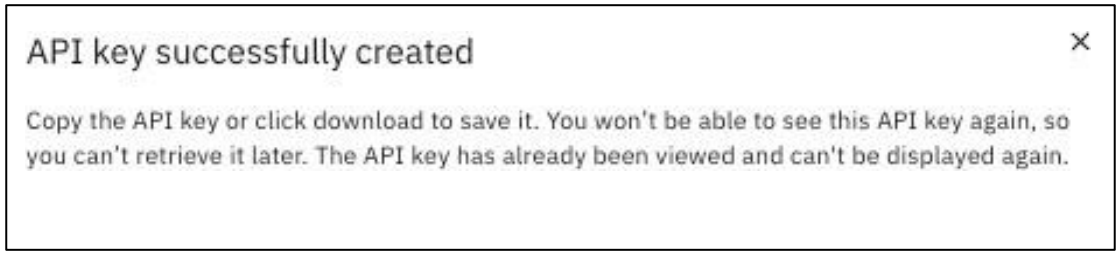
6. Select the blue “eye” icon on the right to look at the key.
7. Copy the key.
8. Download the key.



It is recommended that you download your key. This key will be downloaded to a file called apikey.json. You might want to rename this file to ensure you remember what it is.

If necessary, you can always create another API key.

You might see this message:



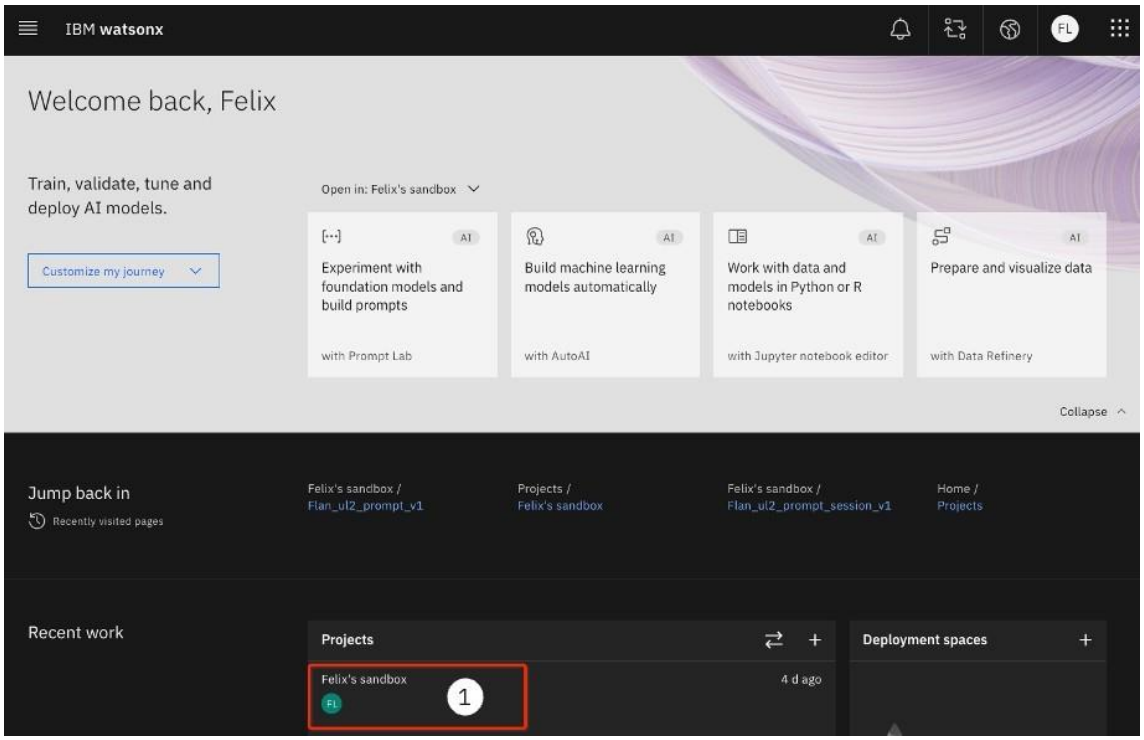
Simply close this window and the API keys window.

Now that you have the key, you are ready to work with the Jupyter notebook. For more information, see Creating an API key.

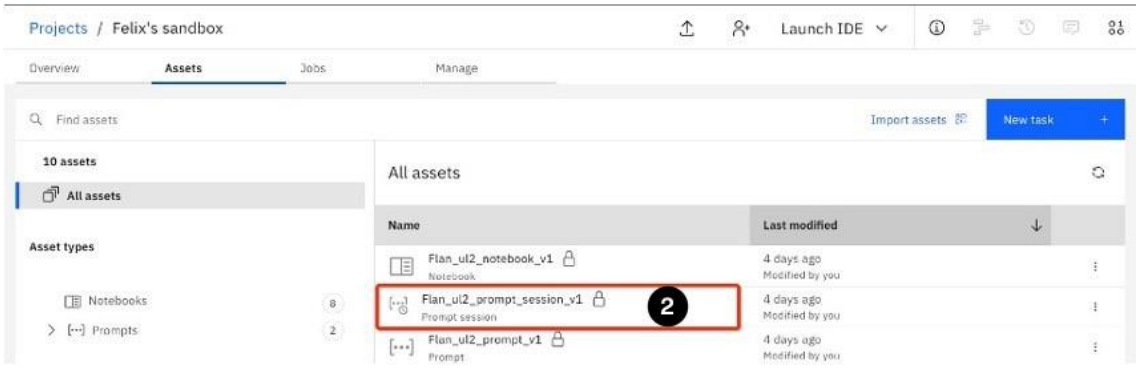
b. Creating a Jupyter notebook from a prompt

If you recall from the prompt and prompt session exercise, there is a third option when you choose to save a prompt, this is what you will do now.

1. Open the watsonx.ai Prompt Lab and scroll down to the Projects and click the project you were using to open it. Most likely it would look like <Your name>’s sandbox.



2. In the list of assets (under the Assets tab), click the Flan_ul2_prompt_session_v1 prompt session.

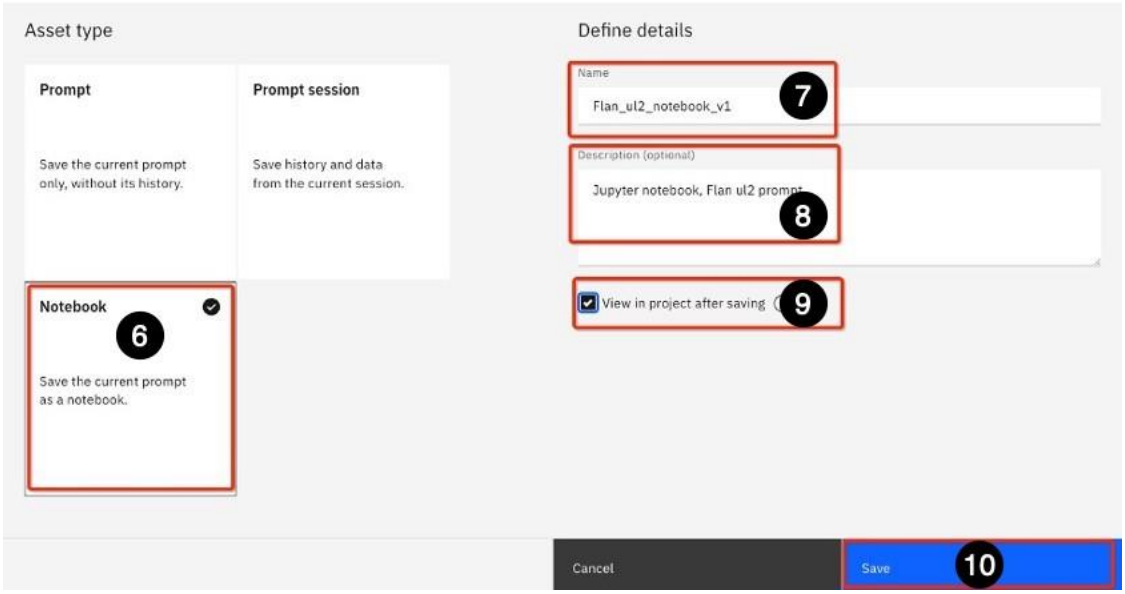


3. Click the clock icon on the left to open the history of this prompt session.

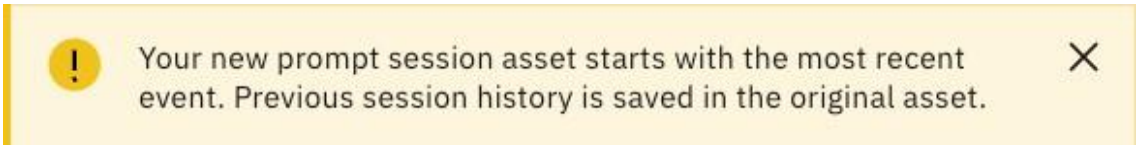
4. If the Now version is not highlighted, click it now.

5. Select Save work, and then select Save as.

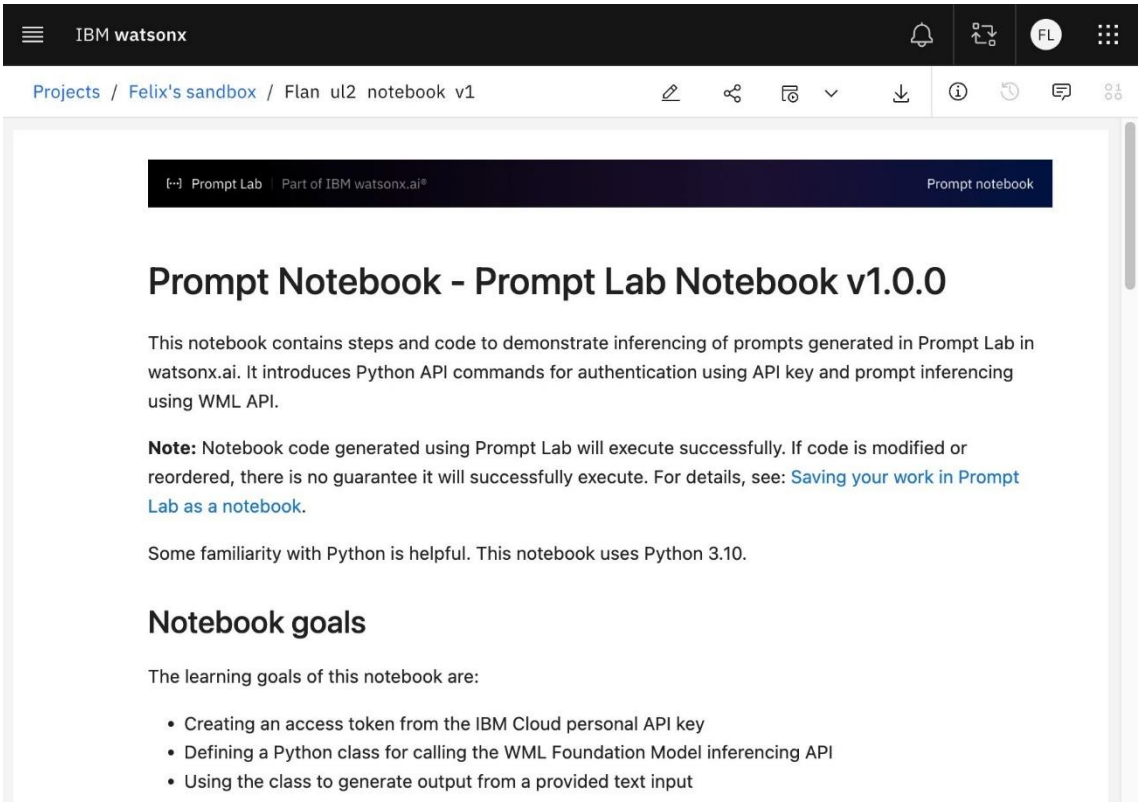
6. Select the Notebook tile and fill in the fields as follows:
7. Use the Name “Flan_ul2_notebook_v1”.
8. Use the Description “Jupyter notebook, Flan ul2 prompt”.
9. Select View in project after saving.
10. Click Save.



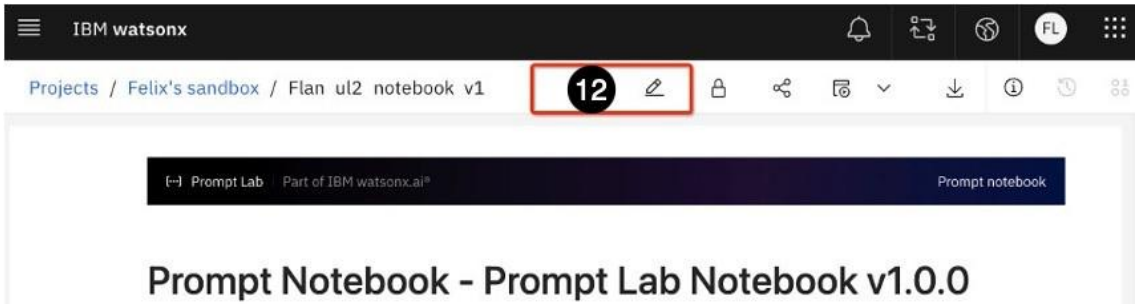
You may see a message like the one below, you can ignore it.



11. Watsonx.ai will open the Jupyter notebook with the content of the prompt.



12. Click the pen icon in the top menu bar to go into edit mode.

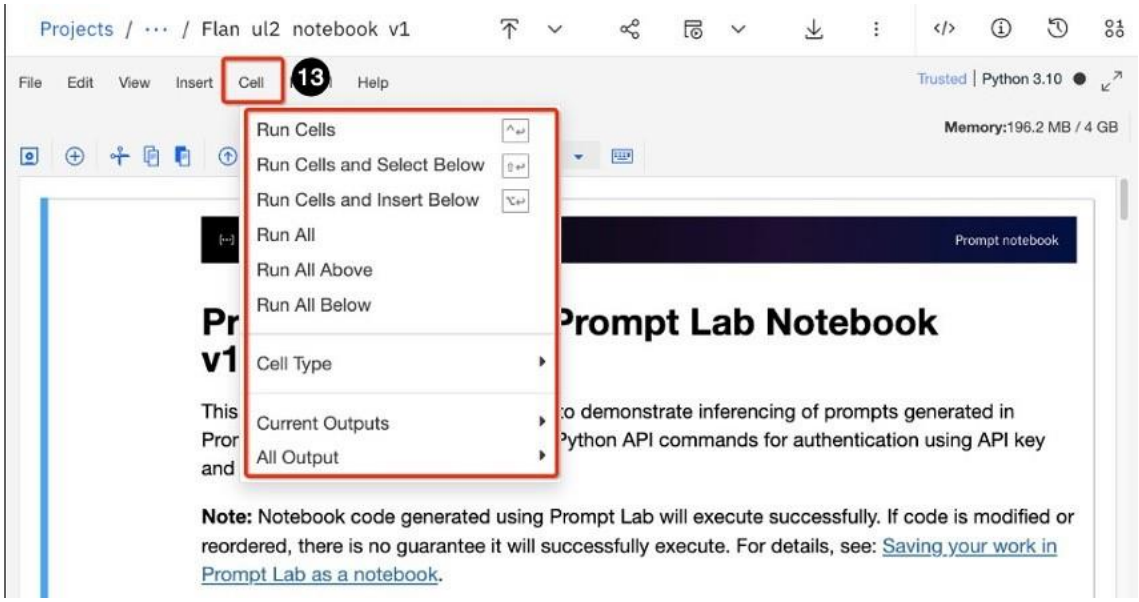


You can now work with the Jupyter Noteboook.

13. Inspect the notebook and the various cells.

The first cell is the setup cell. Watsonx.ai has automatically filled in all the necessary data such as wml_url and sets up an inferencing class that makes a REST API call to the watsonx foundation model to generate output from your input. This part of the notebook is not the focus of this lab so you will not spend more time on this cell.

You can choose to run a single cell or run the entire notebook. Click Cell at the top menu bar and select the right scope for the run operation. You have these options:



14. The next section watsonx API connection defines the credentials necessary to work with watsonx API. When you run this cell, you will be asked to enter your API key.

```
In [*]: from ibm_cloud_sdk_core import IAMTokenManager
        from ibm_cloud_sdk_core.authenticators import IAMAuthenticator, BearerTokenAuthenticator
        import os, getpass

        access_token = IAMTokenManager(
            apikey = getpass.getpass("Please enter your api key (hit enter): "),
            url = "https://iam.cloud.ibm.com/identity/token"
        ).get_token()

           Please enter your api key (hit enter):
```

You need to enter your API key, and “hit enter” as instructed, before executing the next cell. You will be asked on each run to provide your API key. As a workaround, you can modify the code by changing the apikey line:

apikey = getpass.getpass(Please enter your api key (hit enter):”),

TO

apikey = “<your API key>”,

For example, it may look like this (with a made-up key):

```
from ibm_cloud_sdk_core import IAMTokenManager
from ibm_cloud_sdk_core.authenticators import IAMAuthenticator, BearerTokenAu
import os, getpass

access_token = IAMTokenManager(
    apikey = "abcdefghijk1234567890ABCDEFGHIJK_1234567890Z",
    url = "https://iam.cloud.ibm.com/identity/token"
).get_token()
```

After this change, you will not need to provide your key for each run session.

15. The section Defining the model id identifies the model you are using.

Defining the model id

We need to specify model id that will be used for inferencing:

```
model_id = "google/flan-ul2"
```

For the 5 models available, here are the respective names you should use for model_id in a Jupyter notebook:

- flan-ul2-20b model_id=”google/flan-ul2”
- flan-t5-xxl-11b model_id=”google/flan-t5-xxl”
- gpt-neox-20b model_id=”eleutherai/gpt-neox-20b”
- mpt-7b-instruct2 model_id=”ibm/mpt-7b-instruct2”
- mt0-xxl-13b model_id=”bigscience/mt0-xxl”
- starcoder-15.5b model_id=”bigcode/starcoder”
- llama-2-70b-chat model_id=”meta-llama/llama-2-70b-chat”

16. The next section defines the model parameters that you can update. These are the same ones as from the user interface.

- decoding_method greedy or sample
- min_new_tokens 0 or larger
- max_new_tokens positive integer
- temperature 0.0 to 1.0
- top_k 0 to 100
- top_p 0.0 to 1.0
- repetition_penalty 1.0 to 2.0

In this particular example, the following are the initial values used (not all parameters are included):

Defining the model parameters

We need to provide a set of model parameters that will influence the result:

```
parameters = {
    "decoding_method": "sample",
    "max_new_tokens": 100,
    "temperature": 0.7,
    "top_k": 50,
    "top_p": 0.5,
    "repetition_penalty": 1.2
}
```

Once you have put your API key in the notebook, you can easily experiment by changing these runtime configuration parameters and see how they can affect the output.

For example, try changing the values of max_new_tokens in combination with different values of repetition_penalty; and changing values of top_p and top_k.

17. The next section defines the project id, you should not need to touch it.

Defining the project id

The API requires project id that provides the context for the call. We will obtain the id from the project in which this notebook runs:

```
import os

project_id = os.environ["PROJECT_ID"]
```

18. The following section is your prompt_input - your input into the inference engine.

Defining the inferencing input

Foundation model inferencing API accepts a natural language input that it will use to provide the natural language response. The API is sensitive to formatting. Input structure, presence of training steps (one-shot, two-shot learning etc.), as well as phrasing all influence the final response and belongs to the emerging discipline of Prompt Engineering.

Let us provide the input we got from the Prompt Lab:

18

```
prompt_input = """Input:
The following paragraph is a consumer complaint.
The complaint is about one of these options: credit cards, credit reporting,

I bought a GPS from your store and the instructions included are in Spanish,

Output:
The list of issues is as follows:
1) The instructions are in Spanish, not English.
2) The mounting bracket is broken.
3) The information is outdated.

Input:
The following paragraph is a consumer complaint. The complaint is about one c

I called your help desk multiple times and every time I waited 10-15 minutes

Output:
"""
```

- From the model's point of view, everything between the pair of triple double quotes (“””) is the prompt.
- Note that this was the one-shot prompt example you worked on earlier. When you do this (similarly for few-shots), you are simply inserting it before the section that you want to derive results on.
- You can use this notebook to quickly test out a “few-shot” example – simply insert another sample section in the cell.
- Another important item to remember is that here the words Input and Output are not “fixed” and you can change them. You should try changing Output to something different like Summarize or Conclusion. If you do change it, you might also change the Output in the one-shot (in the middle of the text box), or your answer may be surprising.

- Always keep in mind, that this is NOT a human reading the input as natural language. A model sees the input as a series of tokens, and it is calculating probability – changing words in key locations (such as the last word in the prompt) can have interesting impacts.

Appendix

Troubleshooting

1. You get the following error when you try to generate anything from any models from the watsonx.ai Prompt Lab:
   {
   “code”: “invalid_instance_status_error”,
   “message”: “WML instance b7bc6824-e151-4762-b6c5-b2f287308b1b status is not active, current status: Inactive”
   }
   - Reason: Either your WML instance has timed out, or you do not have one.
   - To resolve:
     - Open your project
     - Select the Manage tab
     - If you have a Watson Machine Learning service, remove it
     - Select Associate service
     - Select Watson Machine Learning
     - Select Associate
2. You are progressing with the various labs and sometimes you do not see the same output or panels as the lab suggests.
   - Reason: You might have performed additional exploratory steps (which is not an issue – you should explore) and that might have unintentionally affected how the model may be interpreting the input.
   - To resolve: (essentially do a reset and start from scratch)
     - Select the  icon on the upper left corner
     - Select Home
     - Select Experiment with foundation models and build prompts
     - Start with the step 1 of the section you are working on.

Glossary

Foundation models

are typically built using a specific kind of neural network architecture, called a transformer, which is designed to generate sequences of related data elements (for example, a sentence). These models are trained on extensive datasets and can be fine-tuned with specific data to perform a wide range of tasks, including generative AI and machine learning. IBM's foundation models, such as Granite, IBM Open Source, and Hugging Face Llama 2, are designed to be leveraged with responsibility, transparency, and explainability, ensuring they can be adapted to meet the specific needs of enterprises while maintaining trust and governance.

Generative AI

refers to a set of AI algorithms that can generate new outputs — such as text, images, code, or audio — based on the training data, unlike traditional AI systems that are designed to recognize patterns and make predictions. Sometimes the AI that powers these solutions is referred to as decoders.

Hallucination

is a well-known phenomenon in large language models (LLMs) in which the system provides an answer that is factually incorrect, irrelevant, or nonsensical because of limitations in its training data and architecture; more concerning is the hallucinated answer sounds plausible.

A large language model (LLM)

is a type of machine learning model that has been trained on large quantities of unlabeled text using self-supervised learning and can perform a variety of natural language processing (NLP) tasks (even when that language is a programming language). Output may range from books, articles, social media posts, online conversations, and even code. The architecture of an LLM consists of layers of neural networks that learn to generate language in a way that is similar to how humans use language.

Natural language processing (NLP)

is the technology that gives computers the ability to understand text and spoken words in much the same way human beings can. NLP combines computational linguistics — rule-based modeling of human language — with statistical, machine learning, and deep learning models. These technologies enable computers to process human language in the form of text or voice data and to ‘understand’ its full meaning, complete with the speaker or writer’s intent and sentiment.

Prompt

Input and query that users or programs use to interface with foundation models so they can respond with useful/desirable results. A prompt can be a simple NLP question, or it can be a large body of text. The structure of the prompt is very important in eliciting proper responses from foundation models.

Prompt Engineering

Prompt engineering is the process of crafting prompt text to best effect a given model and parameters.

Zero-shot Prompting

Providing a prompt without any examples to guide the model's response.

One-shot Prompting

Providing a single example to help the model understand the desired output.

Few-shot Prompting

Providing multiple examples to guide the model's response.

Decoder-only model

Models designed explicitly for generative AI use cases; represents the architectures used in GPT-4 and other popular Large Language Models.

Encoder-only model

Models with best cost performance trade-off for non-generative use cases but require task-specific labeled data for fine-tuning.

Encoder-decoder model

Models that support both generative and non-generative use cases. These have the best cost-performance trade-off for generative use cases when the input is large but the generated output is small.

Unstructured data

information that does not have a predefined format or organization. Unlike structured data, which is neatly arranged in tables with rows and columns (e.g., relational databases)

Links to more information

1. How to get to watsonx

This document is an appendix to the main document 'Lab 1 - IBM Prompt Lab' and aims to help you understand more about the **interface of watsonx.ai**. We highly encourage you to read this document where the web-based user interface (UI) is described more thoroughly. It includes the Prompt Lab, the Structured and Freeform interface, model parameter configuration panels, and model information panels.

2. Available Models in Watson X and their specificities

A collection of open source and IBM foundation models are available for inferencing in IBM watsonx.ai. Find foundation models that best suit the needs of your generative AI application and your budget:

https://dataplatform.cloud.ibm.com/docs/content/wsj/analyze-data/fm-models.html?context=wx&locale=en&audience=wdp

3. Watsonx.ai Manual Prompt Engineering Video Tutorial

This video is a simple but detailed guide to Watsonx Prompt Lab: https://www.youtube.com/watch?v=-Kx_C8yVtgw

4. Prompt Lab further explanation

This page will explain with video how to use the Prompt Lab: https://dataplatform.cloud.ibm.com/docs/content/wsj/analyze-data/fm-prompt-lab.html?context=wx&locale=en&audience=wdp